UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

Estudi i Disseny de Personatges Virtuals amb Comportament Intel·ligent Orientat a Objectius i Utilitats

---

Autora: Aina Hernàndez Campaña

Directora: Dra. Inmaculada Rodríguez Santiago

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 27 de juny de 2018

Abstract

Nowdays, studies suggest that the commercial success of a videogame is directly related to the quality and robustness of its AI, which is a challenge for game developers.

This project focuses on the design and implementation of intelligent behaviours for Non-Player Characters (NPC) in a game. Both techniques choose an action that will be performed by the character. There is a wide range of actions available to characters. Specifically, the techniques that will be studied are:

- Goal Oriented Behaviour (GOB): this technique chooses an action based on character's goals. Each goal has a value indicating how satisfied it is, and performing an action can increase or decrease this value for one or more goals. The chosen action will be the one that affects more positively to the world, leaving the major possible number of goals satisfied.

- Utility System: this technique choses an action based on utility values of each action. Each action has associated game state variables. For example, the action *Go to eat* can have associated the variable *character's hunger*. The utility of an action depends on the value of the world variables it has associated.

This work presents results by means of simulations and an educational game called Fracsland, previously developed in another final project. Initially, Fracsland did not incorporate AI in NPCs, with this project they will be capable of making their own decisions.

The chosen NPC to apply the studied techniques is called Lord Barus, the governor of the island. The main idea is to give him an AI to assign quests to the player based on the status of the island. His goals are: keep the population safe, maintain the buildings in decent condition, have the farm animals healthy and possess stocks of food and materials.

Resum

Avui en dia diferents estudis constaten que bona part de l'èxit d'un videojoc depèn de la robustesa i qualitat de la seva Intel·ligència Artificial, fet que suposa un repte per els desenvolupadors.

Aquest projecte es centra en dissenyar comportaments intel·ligents per personatges no jugables (NPCs) d'un videojoc. S'estudien diverses tècniques orientades a objectius i utilitats. Totes dues serveixen per escollir una acció per a que dugui a terme el personatge, i es considera que cada personatge té una sèrie d'accions que pot dur a terme en el joc.

- Goal Oriented Behaviour (GOB): una tècnica per escollir una acció basant-se en els objectius del personatge. Els objectius tenen un grau que indica com de satisfet es troba, i cada acció pot afectar a la satisfacció d'un o varis objectius. L'acció escollida serà la que deixi el major nombre d'objectius satisfets.

- Utility System: una tècnica per escollir una acció a partir dels valors d'utilitat que té cada acció. Cada acció té associades unes variables al joc. Per exemple, l'acció *Anar a menjar* por tenir associada la variable *fam del personatge*. Així, a partir del valor de les variables que influeixen a cada acció, se'ls calcula el seu valor d'utilitat.

Els resultats d'aquest treball és presenten amb simulacions i amb la implementació de la tècnica que dona millors resultats a un joc educatiu anomenat Fracsland, desenvolupat en un projecte de final de grau d'un any anterior. Al començament els personatges no tenien cap tècnica d'Intel·ligència Artificial implementada, i amb aquest treball es dota a un personatge del joc amb Intel·ligència Artificial per a que sigui capaç de prendre decisions amb autonomia.

El personatge escollit s'anomena Lord Barus i és el governador de l'illa. Amb la nova Intel·ligència Artificial serà capaç d'assignar missions al jugador per gestionar i mantenir l'illa en bon estat. Els seus objectius són mantenir la població segura, tenir les edificacions en bon estat, mantenir els animals de granja alimentats i tenir reserves d'aliments i materials.

Agraïments

Vull aprofitar aquest apartat per agrair a totes les persones sense les quals aquest treball no hagués estat possible.

Per començar, vull donar les gràcies a tots els meus familiars i amics pel suport incondicional que m'han donat durant tot aquest temps. En especial a la meva parella, per recolzar-me i encoratjar-me en els moments més difícils.

A l'Aleix Cots, un gran amic sense el qual la carrera no hagués sigut el mateix, fins i tot vam poder iniciar junts aquest projecte.

També al meu company de feina, Gustavo Sandín, per animar-me sempre.

Per últim però no menys important, vull agrair a la meva tutora del projecte Inma Rodríguez Santiago per implicar-se i guiar-me en tot moment, i a la professora Anna Puig Puig, que juntament amb la meva tutora m'han ajudat durant el desenvolupament del treball.

Índex

Índex de figures

1 Introducció

Aquest projecte està emmarcat dins del desenvolupament del motor de la Intel·ligència Artificial de videojocs, amb la finalitat de controlar la presa de decisions i comportament dels Non-Player Characters (NPCs) del joc, es a dir, els personatges no jugables per fer-los semblar més realistes. Existeixen diferents tècniques i algorismes que podrien implementar-se per a aconseguir aquesta finalitat.

Aquest treball ha sigut desenvolupat a partir d'un videojoc realitzat en un Treball de Fi de Grau d'un any anterior, Fracsland[1], el qual és molt prometedor perquè ofereix un ampli ventall de possibilitats en quant a possibles extensions. Els NPCs del joc estaven dotats d'una intel·ligència bastant senzilla o directament nul·la, així que per fer el joc més jugable es faran modificacions en la Intel·ligència Artificial d'aquests personatges. El videojoc serà ampliat utilitzant el motor de joc Unity i programant amb el llenguatge orientat a objectes C#.

Concretament aquest projecte neix amb la necessitat de reforçar la Intel·ligència Artificial dels NPCs del joc amb uns comportaments més realistes fent servir i estudiant dues tècniques diferents: Goal Oriented Behaviour i Utilities.

- Goal Oriented Behaviour (GOB)

  Aquesta tècnica es basa en definir objectius que pot tenir un personatge i programar-lo per a que intenti satisfer-los en la millor manera possible. Cada objectiu té assignat un valor que indica el seu nivell d'importància i té unes accions associades que el satisfaran. Per exemple, si el personatge té com a objectiu més important mantenir-se sa, li interessarà dur a terme accions com menjar verdura o fer esport. El sistema escollirà una acció que satisfaci de millor manera l'objectiu més important en aquell moment.

  És important no confondre aquesta tècnica amb l'anomenada *Goal-Oriented Action Planning* (GOAP) [2]. És un sistema d'Intel·ligència Artificial que permet planificar una seqüència d'accions per satisfer un objectiu. En aquest cas, l'algorisme troba una llista de les accions que s'han de fer per completar un objectiu, a diferència de GOB, que no escull un objectiu a satisfer, sinó que comprova com afecta cada acció per separat al conjunt d'objectius que hi ha i escull una sola acció.

- Utilities

  El concepte principal d'un sistema basat en utilitats és assignar a totes les accions que pot realitzar un NPC una puntuació. Aquest valor, anomenat utilitat, ve donat a partir de les variables que condicionen l'acció. Per exemple, si es si es vol alimentar a un conill, s'haurà de tenir en compte el número de pastanagues que es tenen i com de famolenc es troba el conill, per assignar-li una puntuació o una altra a aquesta acció. El sistema escollirà l'acció que tingui una utilitat major, sent aquesta la més prioritària.

S'ha escollit un dels NPCs més importants de Fracsland, l'alcalde, anomenat Lord Barus, per a dotar-lo amb Intel·ligència Artificial. Aquest personatge és l'encarregat

d'assignar missions al jugador al llarg de la partida. Inicialment, en Lord Barus decidia quina missió assignar en cada moment de forma aleatòria, però aplicant les dues tècniques explicades anteriorment, es vol arribar a aconseguir que l'elecció de cada missió sigui la més convenient tenint en compte l'estat del joc en cada moment. Així, per exemple, si els animals de la granja tenen molta gana, li donarà prioritat a la missió d'anar a alimentar-los.

Aplicant aquestes tècniques d'intel·ligència artificial es vol arribar a obtenir un videojoc més realista i entretingut, on els NPCs tinguin un comportament creïble tenint en compte les limitacions dels equips informàtics on s'espera que es faci servir ja que aquest joc està pensat per ser jugat en ordinador o tauleta.

Al llarg del grau d'Enginyeria Informàtica es cursen varies assignatures que han sigut útils a l'hora de desenvolupar un projecte d'aquest caire. Les més importants a destacar són Enginyeria del Software, on s'ensenya a gestionar tot el procés de creació d'un videojoc des del punt de vista d'una empresa, utilitzant Unity, Factors Humans i Computació, amb la qual es pot entendre millor el punt de vista de l'usuari, i Intel·ligència Artificial, que introdueix els seus conceptes bàsics.

2 Objectius

L'objectiu principal d'aquest projecte és utilitzar Intel·ligència Artificial per a dotar de comportaments intel·ligents als personatges del videojoc Fracsland.

Així, aquest objectiu general es podria desglossar en subobjectius més específics:

1. Estudiar i documentar-se sobre diferents tècniques per la presa de decisions basades en objectius i utilitats.
2. Implementar una aproximació del GOB.
3. Implementar un model de la tècnica basada en utilitats.
4. Estudiar i comparar les dues tècniques.
5. Estudiar el codi de Fracsland per integrar la tècnica que doni millors resultats.

3 Antecedents

3.1 Fracsland

Fracsland és un joc seriós enfocat a nens de 10-12 anys. Els jocs seriosos, o *serious games*[3], són jocs que tenen com a finalitat formar al jugador en algun camp, en aquest cas en l'àmbit educatiu. El seu objectiu principal és ajudar-los a repassar els conceptes bàsics de fraccions d'una forma dinàmica i divertida. És un joc de món obert, on el jugador pot recórrer l'illa on està i realitzar tot tipus d'accions: plantar aliments, alimentar els animals de granja, derrotar enemics, construir edificacions... Encara que el joc té un objectiu principal: completar totes les missions per poder obtenir un vaixell amb el qual sortir de l'illa. Al llarg de l'execució del joc és on apareixen les fraccions.

El jugador pot moure's lliurement per l'illa i pot anar completant missions, que li venen assignades per un personatge del joc: l'alcalde, anomenat Lord Barus. És un NPC del joc que inicialment assigna les missions que té de manera aleatòria.

Altres NPCs estàtics que es poden trobar són els vilatans del poble i el venedor d'armes, el qual també té una interacció senzilla amb el jugador on simplement li mostra la llista d'armes disponibles que té per comprar i el jugador escull. Aquest últim es pot veure a la Figura 1 requadrat en color verd, juntament amb en Lord Barus requadrat en color vermell i el jugador requadrat en color blau.

Un altre tipus de NPCs que es poden trobar són els dinàmics, que comprendrien els animals i enemics de l'illa. Són lleons, llops i pirates, que es van movent per les seves respectives zones i que quan el jugador està a prop o en el seu rang de visió, comencen a perseguir-lo per atacar-lo i treure-li vida.

Figura 1: Captura on es pot veure en Lord Barus, el jugador i el venedor d'armes.

Tal com s'ha esmentat, actualment tots els NPCs tenen implementada una lògica molt senzilla. Per tant, una bona extensió per fer el joc més jugable és dotar a aquests NPCs amb una Intel·ligència Artificial per la seva interacció amb el jugador.

3.2 Tècniques d'IA en jocs basats en utilitats

Un dels sistemes que més s'han fet servir durant els últims temps per aconseguir aquest tipus de realisme pels NPCs dels jocs són els *Behaviour Trees*. Encara es pot considerar el sistema dominant en la indústria de desenvolupament de videojocs, tot i que recentment s'està veient que està quedant antiquat, ja que cada cop es demanen Intel·ligències Artificials més sofisticades que permetin als NPCs tenir comportaments realistes sota qualsevol possible estat del joc.

Els *Behaviour Trees* treballen a partir d'una Màquina d'Estats Finita (FSM), amb la qual es defineixen estats i condicions sobre com passar d'un estat a un altre. A partir dels estats i condicions s'arriben a diferents comportaments o *behaviours*, per tant l'algorisme haurà d'evaluar l'arbre per saber quin comportament és més adient per fer en aquell moment. És pot veure un exemple senzill d'un *Behaviour Tree* a la Figura 2.

Els *Behaviours Trees* tenen l'avantatge que dissenyar i estructurar amb aquest tipus de jerarquia ajuda a tenir un bon esquema visual del comportament que es segueix. Malgrat això, apareixen desavantatges a mesura que un *Behaviour Tree* es fa gran i complex, ja que el cost d'evaluar tot un arbre per escollir el comportament que fer a continuació pot ser enorme. També es pot complicar molt el disseny de l'arbre, i encara que s'hagin intentat fer millores introduint *subtrees*, aquests problemes a la llarga persisteixen.

A més, cada cop el nombre de NPCs dels videojocs va en augment, així doncs el repte està en trobar una tècnica que sigui senzilla de definir però amb la qual es puguin aconseguir comportaments complexos. Aquesta secció explica el funcionament dels dos algoritmes principals fets servir per desenvolupar aquest projecte que solucionen els principals problemes que tenen avui en dia els *Behaviour Trees*.

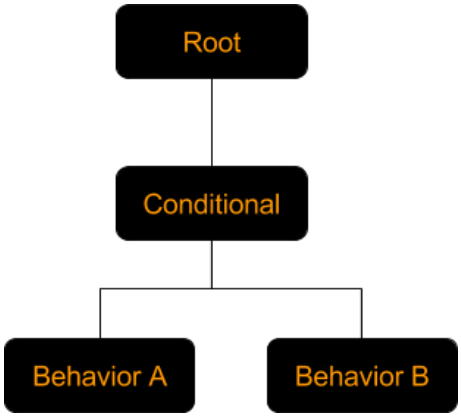


Figura 2: Exemple de *Behaviour Tree* [4]. Comença amb el node arrel que crida al node que conté la condició. Segons el resultat d'aquesta, s'escollirà entre el comportament A o B

3.2.1 GOB

Goal-Oriented-Behaviour és una tècnica definida per Millington [5] per escollir una acció basant-se en els objectius del personatge. Per això és necessari definir les accions i els objectius del NPC.

Un personatge pot tenir un o varis objectius, que es podrien definir també com a desitjos. Cada objectiu té un nivell d'importància, també anomenat *insistència*, que es representa amb un valor. Tenir una insistència a 0 vol dir que l'objectiu està totalment satisfet. Quant més gran sigui aquest valor, més influenciarà el comportament del personatge.

Un cop es defineixen els objectius, s'han de buscar les accions que els satisfacin. Per escollir quina acció convé més fer en un moment determinat, s'ha de comparar amb cada objectiu per veure com li influenciaria. Per exemple, l'acció "Sortir a córrer", pot incrementar la salut d'un personatge, però decrementarà la seva energia. Aquesta comparació també és quantifica numèricament amb un valor anomenat insatisfacció o *discontentment*, que es pot definir en com de insatisfet quedarà un personatge després de realitzar l'acció en qüestió. Per tant, s'escollirà l'acció amb el valor de *discontentment* més petit, ja que voldrà dir que és l'acció que deixa els objectius del personatge més satisfets.

3.2.2 Utility system

Els sistemes que implementen aquesta tècnica es basen en calcular un valor d'utilitat per cada acció que pot fer un personatge i escollir l'acció amb el valor més alt. Per això requereix definir les accions del joc.

Per calcular les utilitats, s'ha de tenir en compte com està l'estat del joc (aquestes inclouen les d'estat del personatge) en aquell moment i totes les variables que influencien a una acció. Així, per exemple, si disposem de l'acció "Anar a dormir", la seva utilitat es calcularà a partir del nivell de son del personatge i de si es disposa

d'un llit o no.

Quan es calculen els valors d'utilitat, és important mantenir una consistència, ja que com s'han de comparar les unes amb les altres per arribar a una decisió final, totes han d'estar en la mateixa escala. Una manera d'aconseguir és normalitzar els valors.

Un exemple de joc conegut que utilitza aquesta tècnica és el popular videojoc de simulació *The Sims* (Figura 3). En aquest joc, els personatges tenen unes variables d'estat que influeixen directament en quina acció decideixen fer a continuació. Així, si un personatge té una variable d'estat per indicar el nivell de gana, si aquesta té un nivell baix, l'acció d'anar a menjar es farà prioritària.

Figura 3: Versions del videojoc *The Sims* que existeixen.

Per dissenyar la presa de decisions dels seus personatges, s'utilitzen sistemes de lògica difusa [6]. Les variables d'estat dels personatges que es tenen en compte són les següents:

- Variables físiques: fam, higiene, bufeta i comoditat.
- Variables emocionals: energia, diversió, socialització i espai.

Cada de les anteriors variables es mesura en un rang comprès entre -100 i 100, i es mapegen al seu corresponent valor d'utilitat utilitzant *response curves*. Van des de funcions lineals (com pel cas de l'energia) fins a polinomials (com en el cas de la fam). Així, la seva Intel·ligència Artificial implementada per modelar el comportament dels personatges es basa en trobar els objectes (i dur a terme una acció específica amb ell) per satisfer les necessitats més prioritàries, és a dir, les variables que donen uns valors d'utilitat més alts amb les seves respectives funcions.

Un altre videojoc famós que utilitza el Sistema d'Utilitats per la presa de decisions dels seus NPCs és *Killzone 2* (Figura 4), que ha estat considerat com uns dels videojocs amb millor Intel·ligència Artificial dissenyada.

Figura 4: Caràtula del videojoc *Killzone 2*.

Respecte als *Behaviour Trees*, aquestes les tècniques de GOB i Sistema d'Utilitats presenten diverses avantatges:

1. Són simples de definir: tant els valors *discontentment* en la tècnica GOB, com els valors d'utilitat en el Sistema d'Utilitats, es poden definir en llenguatge natural, fet que ajuda als programadors a entendre's amb els dissenyadors del joc ja que no cal utilitzar conceptes informàtics o matemàtics.

2. Són fàcilment extensibles: per modificar els valors de *discontentment* o d'utilitat d'una acció és tant fàcil com afegir més variables a les fórmules que les calculen, no com en el cas dels *Behaviour Trees*, que s'haurien de modificar estats i les possibles transicions entre ells.

3. Posseeixen una millor qualitat: la simplicitat en el disseny de la IA permet reduir dràsticament els possibles bugs i augmentar la productivitat, ja que això comporta que també siguin més fàcils de debugar. Aquesta simplicitat també els permet tenir un gran nombre d'accions per un gran nombre de NPCs, a diferència dels *Behaviour Trees*.

4 Anàlisi

4.1 Anàlisi de l'actual versió de Fracsland

Com s'ha explicat en seccions anteriors, Fracsland és un joc de rol de món obert on el personatge por moure's lliurement per l'entorn per explorar-lo i anar completant les missions donades pel joc, concretament pel personatge *Lord Barus*. El joc comença explicant al jugador que el seu personatge ha naufragat i ha anat a parar a una illa, i el seu objectiu principal és aconseguir construir-se un vaixell per marxar. Per fer-ho, necessita un conjunt de peces, que les obtindrà a mesura que vagi completant les missions.

Aquesta illa està composta per quatre escenes, cadascuna amb una temàtica diferent i que compta amb diverses missions:

- Fracstown

  És la zona del poblat. Aquí es troben dos NPCs amistosos: en Lord Barus, que és l'alcalde del poble i qui assigna les missions al jugador, i el venedor d'armes. Tots dos estan quiets esperant a que el jugador interactuï amb ells. És la escena inicial del joc on el jugador comença la partida.

- Little Beach

  Aquesta escena és una platja amb palmeres i lleons. Els lleons són NPCs enemics que romanen quiets fins que el jugador s'acosta a ells, que llavors comencen a perseguir-lo i atacar-lo. També es troba un pont destruït que el jugador haurà de construir en una de les missions fent servir fusta que pot aconseguir talant les palmeres d'aquesta zona.

- FracFarm

  En aquesta zona hi ha una granja amb un hort per plantar verdures i animals per alimentar: un conill, una vaca i un pollastre. Tots tres són NPCs neutrals que estaran esperant sense moure's a que el jugador els alimenti. El jugador pot plantar vegetals a l'hort o alimentar als animals de manera lliure o quan li vé donat per una missió.

- Frozen Woods

  La quarta escena del joc consisteix en un bosc nevat on es troben els NPCs pirates i llops, que són enemics. Es van movent per la zona i quan s'acosten al jugador, comencen a perseguir-lo per atacar-lo. Un dels pirates és el Capità John, a qui el jugador haurà de derrotar en una de les missions del joc.

A la Figura 5 es poden veure imatges de les quatre zones de Fracsland.

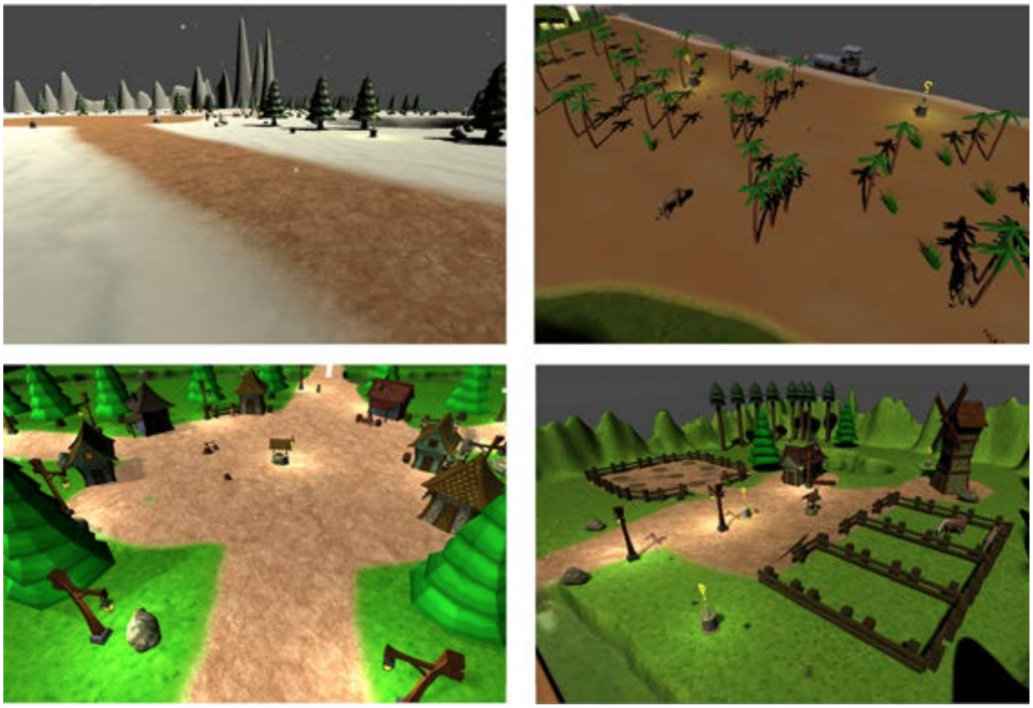


Figura 5: Frozen Woods, Little Beach, FracsTown i FracFarm.

4.2 IA a modelar per a un personatge de Fracsland

Tenint en compte els NPCs amb els que compta el joc, el més adient per aplicar-li una Intel·ligència Artificial per prendre decisions és l'alcalde del poble, en Lord Barus, el personatge que assigna les missions al jugador.

Al principi es va considerar també el venedor d'armes, però les possibilitats que oferia eren més reduïdes, ja que la venta d'armes cobreix una part petita del joc, mentre que les missions tenen un pes molt més gran i important dins d'una partida.

Una missió o *quest* és un esdeveniment important dins del joc on el jugador ha de seguir uns passos i interaccionar amb uns elements concrets per resoldre-la. Actualment totes les missions estan associades amb la resolució de fraccions. Les missions que pot assignar en Lord Barus són les següents:

- Construir el pont que es troba a la platja:

  Per dur a terme aquesta missió, el jugador ha d'anar a la zona de *Little Beach*, recollir fusta (per fer-ho s'han de talar palmeres amb una destral que pot comprar a la botiga del venedor d'armes) i arribar fins on està el pont destruït. Apareixerà un nou repte amb fraccions i si es completa correctament, el pont es construirà i s'haurà completat la *quest*.

- Alimentar al conill de la granja:

  Per completar aquesta missió correctament, el jugador ha de moure's fins a la zona de *FracFarm*, i primer de tot plantar una pastanaga a l'hort. Apareixerà

un nou repte amb fraccions i si es completa correctament, el conill vindrà i el jugador l'haurà de tocar per alimentar-lo. Així quedarà ben alimentat i s'haurà completat la quest.

- Alimentar a la vaca de la granja:

  Aquesta missió és realitza de la mateixa manera que la d'alimentar el conill, solament canviant pastanaga per herba i l'animal per la vaca.

- Alimentar el pollastre de la granja.

  Aquesta missió és realitza de la mateixa manera que la d'alimentar el conill, solament canviant canviant pastanaga per herba i l'animal pel pollastre.

- Derrotar el capità pirata que atemoreix a la població:

  Per dur a terme aquesta missió, el jugador ha de desplaçar-se fins la zona *Frozen Woods* i arribar fins on es troba el pirata. Apareixerà un nou repte amb fraccions i si es completa correctament, derrotarà al pirata i s'haurà completat la quest.

Com que Fracsland és una illa amb un poblat, se li pot afegir la característica de que en Lord Barus assigni al jugador les missions més importants en cada moment per tenir al poble satisfet. Com és l'alcalde, li interessa mantenir la seguretat del poble, els edificis en bon estat, reserves d'aliments... Aquests serien els seus objectius.

Les missions que en Lord Barus assigna al jugador correspondrien a les accions disponibles en cada moment. Tenint en compte que per fer una missió es necessiten fer varies accions, es poden desglossar i el nombre total d'accions augmenta. Així, el món no és tant simple i l'algorisme per escollir quina acció fer en cada moment dona resultats més realistes. Es proposa que la llista d'objectius amb les seves respectives accions que els satisfan sigui la següent:

- Objectiu *GoalSeguretat*: Mantenir la seguretat de la població.
  - Acció: Derrotar a un lleó
  - Acció: Derrotar al capità pirata
- Objectiu *GoalAlimentar*: Mantenir els animals de la granja ben alimentats.
  - Acció: Alimentar el pollastre
  - Acció: Alimentar la vaca
  - Acció: Alimentar el conill
- Objectiu *GoalConstruir*: No tenir edificacions en runes.
  - Acció: Construir la tanca
  - Acció: Construir el pont
- Objectiu *GoalStocks*: Tenir reserves d'aliments i materials pel poble.

- Acció: Plantar herba
- Acció: Plantar pastanaga
- Acció: Recollir fusta

4.3 Escenaris a modificar

Per implementar aquesta proposta bàsicament s'ha de canviar la lògica d'en Lord Barus introduint els nous algorismes per la seva presa de decisions. Concretament, hi ha un mètode on s'inicialitzen les missions i s'escullen aleatòriament per mostrar-les al jugador. S'haurà de canviar aquest mètode per a que esculli l'acció més adient basant-se en l'estat actual del joc i segons com es trobin els objectius del personatge.

4.4 Requeriments tecnològics

El joc està pensat per ser jugat en ordinador, i com que Unity té uns requisits mínims per poder executar-lo no es necessiten uns grans requeriments en el sistema.

A nivell de software requereix que el sistema operatiu sigui de Windows Vista cap endavant, Mac OS X 10.9 cap endavant o Ubuntu 12.04 cap endavant, depenent de si es tracta d'un Windows, Mac o Linux respectivament. També pot ser SteamOS.

En quant a hardware requereix una targeta de vídeo amb capacitat per DX10 (shader model 4.0) i que la CPU sigui compatible amb el conjunt d'instruccions SSE2.

Es pot trobar més informació sobre els requeriments que es necessiten tant a nivell de desenvolupadors com per executar el joc en la següent adreça:

`https://unity3d.com/es/unity/system-requirements`

5 Disseny de la IA

Per provar diferents tipus de comportaments i preses de decisions pel personatge seleccionat, en Lord Barus, s'han fet proves i simulacions amb les dues tècniques esmentades anteriorment: Goal Oriented Behaviour i Utility System.

En totes dues aproximacions, el primer que s'ha fet ha sigut implementar l'algorisme adient i fer proves simulant un número N d'iteracions per veure en quin estat quedava el joc a llarg termini. En tot moment en les simulacions s'ha considerat un perfil de jugador expert, que sempre du a terme l'acció assignada correctament sense errar.

Un cop comparats els resultats, s'ha aplicat al Lord Barus la tècnica que proporciona uns resultats més òptims per a que finalment mostri al jugador la missió que ha de fer quan aquest li demani. Aquest estudi comparatiu es podrà veure en el Capítol 7 (Resultats i Simulacions) d'aquest treball .

5.1 Diagrama de Classes

Per tal d'implementar les dues tècniques, explicades en els següents subapartats, primer de tot s'ha fet un disseny de les classes necessàries, que ha acabat sent comú per ambdues. Difereix en el mètode que cal cridar per calcular quina acció escollir en aquell moment.

La Figura 6 mostra les noves classes que s'han creat i la classe que s'ha hagut de modificar. Totes les accions que hereten de la classe `ActionGOB`, tenen els mateixos mètodes, però per estalviar espai només s'ha especificat en una (`FeedCowAction`) i s'ha posat punts suspensius en la resta.

La funcionalitat de cadascuna d'aquestes classes són les següents:

- `ActionGOB`: Classe abstracta que conté tota la informació necessària d'una acció. D'ella hereten totes les accions disponibles del joc. Els mètodes més importants que té son:
  - `getGoalChange`: Retorna un valor indicant com afecta la pròpia acció a l'objectiu passat per paràmetres.
  - `getDuration`: Retorna un valor indicant el temps que triga en dur-se a terme l'acció.
  - `getQuestInformation`: Retorna un objecte `QuestInformation` amb tota la informació de l'acció. Només està implementada en les accions que són les missions actualment disponibles en el joc: `FeedCowAction, FeedRabbitAction, FeedChickenAction, DefeatPirateAction` i `BuildBridgeAction.`
- `GoalGOB`: Classe abstracta que conté tota la informació necessària d'un objectiu. D'ella hereten tots els objectius que hi ha en el joc. Els mètodes i atributs més importants que té son:

  - `name`: nom de l'objectiu.
  - `value`: valor de satisfacció (importància) que té en un moment determinat l'objectiu.
  - `getGoalChange`: Retorna un valor indicant com es modifica la satisfacció de l'objectiu en una unitat de temps.
  - `calculateChangeThroughTime`: Retorna un valor indicant com ha canviat la importància de l'objectiu quan a passat un temps concret

- `ActionChooser`: classe que conté els algorismes de les dues tècniques. Amb el mètode `chooseGOB` es crida a l'algorisme per la tècnica Goal Oriented Behaviour i amb el mètode `chooseUtilities` es crida a la tècnica que fa servir Utilitats.

- `ActionSimulator`: Classe que s'encarrega de gestionar tota la simulació del món fent N iteracions. És a dir, inicialitza totes les variables necessàries i per cada iteració crida al mètode corresponent de la classe `ActionChooser` per obtenir l'acció a realitzar i actualitza el món com si aquella acció s'hagués realitzat satisfactòriament.

- `QuestDetailArchive`: Dins de l'execució del joc, aquesta és la classe que obté l'acció necessària a fer quan el jugador interacciona amb en Lord Barus. Aquesta classe ja existia, només ha calgut modificar el mètode `chooseQuest()`. Aquest mètode mostrava al jugador 3 missions, una de cada zona de Fracsland escollida de forma aleatòria, per a que ell mateix escollís quina fer. Ara crida al mètode corresponent de la classe `ActionChooser` per trobar l'acció més adient. Aquest mètode ha sigut implementat un cop les proves amb la simulació han conclòs i s'ha escollit quina és la tècnica mes adequada per utilitzar a Fracsland.

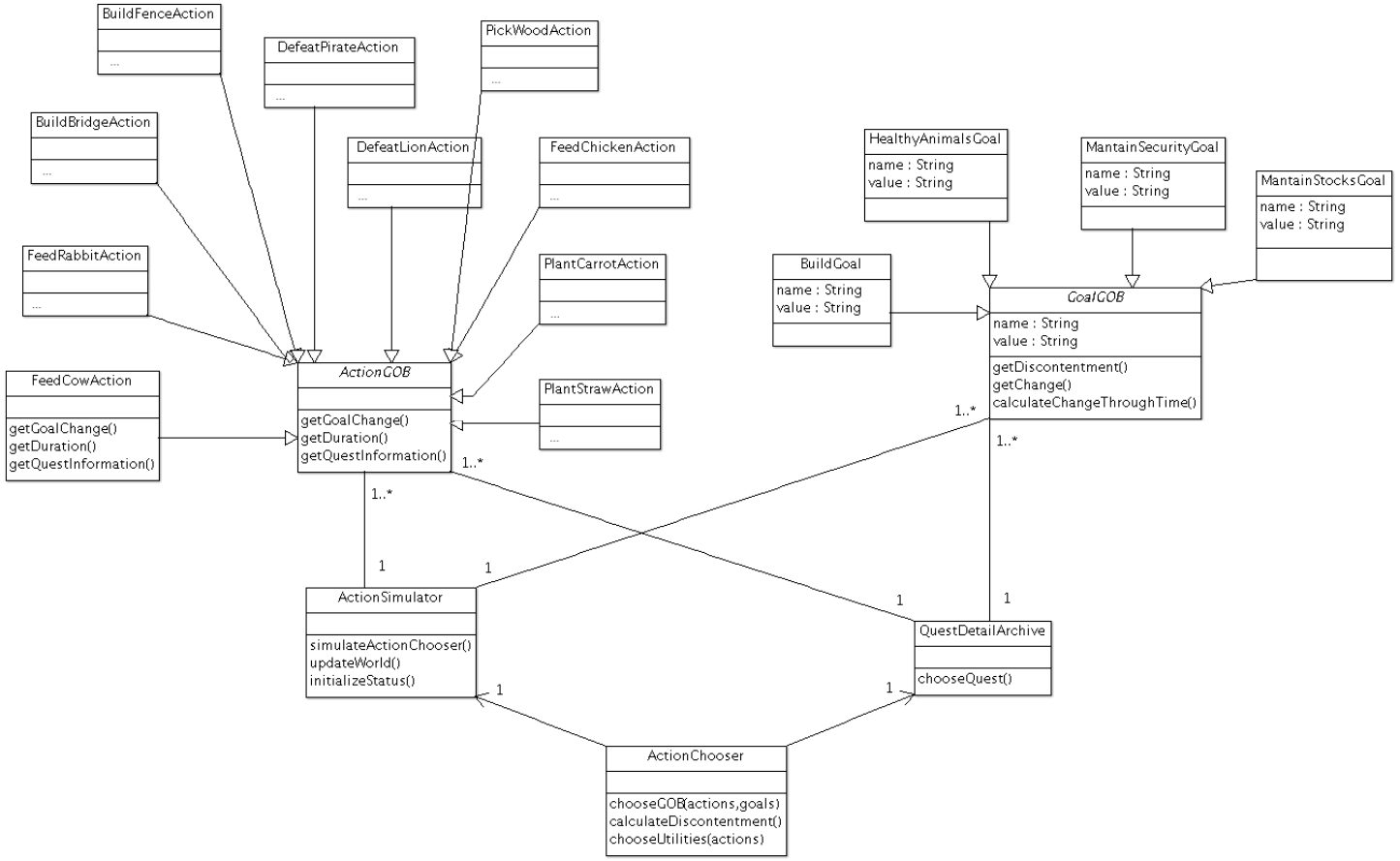


Figura 6: Diagrama de Classes.

També s'han definit les *variables d'estat del joc* que influiran a les accions i objectius del personatge. Es troben a les classes `IslandMonitor` i `IslandWharehouse`. No apareixen al diagrama de classes perquè s'ha considerat que no són tant rellevants, només guarden la informació sobre aquestes variables. També s'indica l'interval de valors que poden prendre.

- Rabbit hunger: percentatge de gana que té el conill. [0-100].
- Cow Hunger: percentatge de gana que té la vaca. [0-100].
- Chicken Hunger: percentatge de gana que té el pollastre. [0-100].
- Num Straw: unitats d'herba que es té a l'inventari. [0-40].
- Num Carrots: número de pastanagues que es té a l'inventari. [0-40].
- Wood ammount: unitats de fusta que es té a l'inventari. [0-40].
- Max Stock: constant que indica el màxim d'unitats que es pot tenir de cada producte. [40].
- Num Lions: número de lleons vius. Inicialment n'hi han 8. [0-8]
- Status Captain: flag que indica si el capità pirata ha sigut derrotat (0) o encara està viu (1).
- Status Fence: flag que indica si la tanca s'ha construït (1) o encara es troba en runes (0).
- Status Bridge: flag que indica si el pont s'ha construït (1) o encara es troba en runes (0).

5.2 Aproximació 1: Goal-Oriented-Behaviour (GOB)

Aquesta és la tècnica basada en la proposta de Millington [5]. Defineix un algorisme [7] per a trobar l'acció més adient donats uns objectius i l'estat actual del món.

S'ha de tenir en compte que cada acció influeix a cada objectiu d'una manera diferent, per exemple, l'acció *Construir Pont* és positiva per l'objectiu de *No tenir edificacions en runes*, però afecta negativament al de *Mantenir stocks*, en aquest cas de fusta, ja que per a construir el pont es necessiten 3 unitats d'aquest material. Per tant, s'haurà d'analitzar per a cada acció com influeix a cada objectiu i retornar la que tingui una afectació global més positiva (o dit d'una altra manera, menys negativa).

Millington defineix el concepte de l'afectació global com a insatisfacció o *discontentment* d'un personatge. Els valors d'importància dels objectius que són alts deixen al personatge insatisfet, per tant l'algorisme escollirà l'acció que més minimitzi el valor de *discontentment* del personatge.

Una altra ampliació d'aquest algorisme és tenir en compte el temps: no totes les accions triguen el mateix temps en completar-se, i això és un factor important a considerar a l'hora d'escollir-ne una. Pel cas concret de Fracsland, les missions són les accions més llargues, ja que es triga a prop de 5 minuts en completar-les, mentre que altres accions com les de recol·lecció de vegetals i fusta es triga al voltant d'un minut en dur-les a terme. Per tant, a en Lord Barus potser li convé assignar al jugador una acció més curta primer per satisfer un objectiu prioritari, abans d'assignar-li una missió llarga.

S'han definit 4 tipus de temps, de menys a més llarg, en el qual classificar les diferents accions:

1. Recollir fusta (tocar una palmera), Derrotar lleó (tocar varies vegades a l'animal)
2. Alimentar al conill, Alimentar la vaca, Alimentar el pollastre (anar fins a l'animal i arrossegar la verdura), Plantar herba, Plantar pastanaga (esperar 1 minut a l'hort fins que creixin)
3. Derrotar al capità pirata, Construir el pont, Construir la tanca (són les missions, primer s'han de recollir fraccions)

En el càlcul proposat per Millington, la variació d'un objectiu en el temps se li suma al valor de *discontentment* i el valor de l'objectiu.

En definitiva, aquest algorisme recorre la llista d'accions i per a cada una fa el calcul del *discontentment* de cada objectiu. Es sumen els valors i s'escull l'acció que tingui el valor de *discontentment* menor. En el cas que hi hagin dues o més accions on aquest valor sigui igual, s'escull la primera acció que s'hagi calculat. Una altra estratègia possible podria ser escollir l'acció de forma aleatòria.

Per a calcular el valor de com una acció afectarà a un objectiu, és a dir el *discontentment*, s'ha provat amb dues aproximacions diferents: utilitzant valors discrets o bé utilitzant valors donats per funcions contínues.

5.2.1 Valors dels objectius

Els valors que prenen els objectius en un moment determinat també venen donats per les variables del món esmentades a la secció anterior. Tots són percentatges, pel que el rang de valors que poden prendre van de 0 a 100. Quan més gran sigui el valor, més prioritari serà satisfer l'objectiu.

- Mantenir la seguretat de la població.

  El valor d'aquest objectiu ve donat per una ponderació amb l'estat del capità pirata i el percentatge de lleons vius.

  $$goalSeguretat = 70 \times captainState + 30 \times (1 - \frac{numLions}{totalLions})$$

  Inicialment té un valor de 100, indicant que no hi ha seguretat ja que tots els lleons i el capità pirata estàn vius.

- Mantenir els animals de la granja alimentats.

  El valor que pren aquest objectiu és la mitjana de la gana dels animals de granja.

  $$goalAlimentar = \frac{rabbitHunger + cowHunger + chickenHunger}{numAnimals}$$

  Inicialment té un valor de 0, ja que cap animal té gana.

- No tenir edificacions en runes

  El valor d'aquest objectiu és la proporció d'edificis no construïts respecte els totals.

  $$goalConstruir = 100 - (50 \times bridgeState + 50 \times fenceState)$$

  Inicialment té un valor de 100, ja que tots dos edificis es troben en runes.

- Tenir reserves d'aliments i materials pel poble.

  El valor que pren aquest objectiu es calcula fent la mitjana de tots els stocks que interessen per a les accions.

  $$goalStocks = 100 - \frac{woodStock + carrotStock + strawStock}{maxStocks}$$

  Inicialment té un valor de 100, ja que no hi ha reserves de cap aliment.

5.2.2 Accions: Valors discrets

En la primera aproximació, es comprova com afecta cada acció a cada objectiu existent, i depenent de l'estat de les variables del món, retorna un valor o un altre. Com que no hi ha una única solució correcta, els valors han sigut ajustats manualment per aconseguir uns *discontentments* amb sentit a l'hora d'escollir accions. A continuació es detalla per a cada acció, quins valors discrets se li han assignat depenent de quin objectiu analitza. És comú per a totes les accions que, si a l'acció no li afecta un objectiu, retornarà 0 per a que no sumi ni resti valor al *discontentment* final.

- Construir pont, Construir tanca:

  Les accions de construir edificacions afecten al *GoalConstruir* positivament, ja que augmenta el nivell d'edificis en bon estat, i també afecten al *GoalStocks* negativament, ja que per construir les edificacions s'ha de gastar fusta.

  Amb un exemple senzill, si tenim que *GoalConstruir* actualment té un valor d'insistència de 5 i *GoalStocks* té un valor d'insistència de 4, aquestes accions els afectarien de la següent manera:

  - Acció Construir Pont (*GoalConstruir* - 3 , *GoalStocks* + 1)
  - Acció Construir Tanca (*GoalConstruir* - 3 , *GoalStocks* + 1)

Si es realitzés una d'aquestes accions, els valors dels objectius passarien a ser: *GoalConstruir* = 2, *GoalStocks* = 5. És a dir, s'hauria satisfet més l'objectiu *GoalConstruir*, i el seu valor d'insistència seria més baix i per tant menys prioritari que *GoalStocks*.

A més, segons com es trobin les variables del món en aquell moment, variarà el valor de *discontentment* que retorna. Pel *GoalConstruir*, si l'edificació es troba en runes, retornarà un nombre molt petit, mentre si ja es troba construïda, retornarà un nombre molt gran per a que sigui impossible que l'acció surti escollida.

Com s'ha esmentat, les accions relacionades amb la construcció també afecten al *GoalStocks*. Si es tenen menys de 3 unitats de fusta (la quantitat que es necessita per construir-les), retornarà un nombre molt gran de *discontentment* per a que no sigui possible escollir aquesta acció, ja que seria impossible de dur-la a terme.

- Derrotar a un lleó:

  Aquesta acció afecta al *GoalSeguretat* positivament, ja que al matar un lleó augmenta el nivell de seguretat de la població.

  Seguint l'exemple mostrat anterior, aquesta acció afectaria al objectiu esmentat de la següent manera:

  – Acció Derrotar Lleó (*GoalSeguretat* - 1)

  A part, pel càlcul de *discontentment*, si no hi han lleons vius, retornarà un nombre molt elevat per a que no es pugui escollir l'acció. Pel contrari, si tots els lleons estan vius per retornarà un valor molt petit, o bé si ja queden menys de la meitat retornarà un valor una mica més gran.

- Derrotar al capità pirata:

  Afecta al *GoalSeguretat* positivament, ja que al derrotar-lo augmenta el nivell de seguretat de la població.

  – Acció Derrotar Capità Pirata (*GoalSeguretat* - 3)

  Com es pot veure, aquesta acció satisfà més el *GoalSeguretat* que l'acció de Derrotar un Lleó. Això és per que el capità pirata atemoreix molt a la població i suposa una amenaça més gran que no pas els lleons per separat.

  En aquest cas, si el capità està viu retorna un valor de *discontentment* molt petit per a que sigui una acció amb moltes possibilitats de ser escollida. En cas contrari, retorna un valor molt elevat indicant que no és possible realitzar l'acció.

- Alimentar el pollastre, Alimentar la vaca, Alimentar el conill:

  Les accions d'alimentar animals afecten a *GoalAlimentar* positivament, ja que fa baixar la gana de l'animal en qüestió. També afecten al *GoalStocks* negativament, ja que redueixen el nombre de reserves del vegetal amb el que s'alimenta l'animal pertinent.

- Acció Alimentar Pollastre (*GoalAlimentar* - 2, *GoalStocks* + 1)
- Acció Alimentar Vaca (*GoalAlimentar* - 2, *GoalStocks* + 1)
- Acció Alimentar Conill (*GoalAlimentar* - 2, *GoalStocks* + 1)

Per aquestes tres accions, per calcular el valor de *discontentment* del *GoalAlimentar*, si no hi han stocks del vegetal que necessita l'animal, retornarà un valor molt elevat perquè no es podrà realitzar l'acció. En cas contrari, retorna valors més petits depenent si l'animal es troba entre el 40, 60 o més percentatge de gana.

Aquestes accions calculen el valor de *discontentment* del *GoalStocks* de la següent manera: si es té bastanta quantitat de reserves del vegetal que es necessita, retorna un nombre més petit per incentivar a l'acció a que surti escollida, indicant que si es gasten reserves d'aquell aliment no passa res perquè la quantitat guardada és gran.

- Plantar herba, Plantar pastanaga:

  Les accions de plantar vegetals afecten positivament al *GoalStocks*, ja que augmenten les reserves del vegetal en qüestió. També afecten negativament al *GoalAlimentar*, ja que per poder alimentar un animal es necessita tenir el vegetal pertinent i gastar una unitat.

  - Acció Plantar Herba (*GoalStocks* - 2, *GoalAlimentar* + 1)
  - Acció Plantar Pastanaga (*GoalStocks* - 2, *GoalAlimentar* + 1)

  Pel *GoalStocks*, aquestes dues accions calculen el valor de *discontentment* de la següent manera: quanta més quantitat es tingui del vegetal en qüestió, més gran serà el valor que retorni. Si no es té cap unitat, aquest valor és molt petit perquè es converteixi en una acció prioritària.

  Com s'ha esmentat, aquestes dues accions també afecten al *GoalAlimentar*. Per calcular el valor de *discontentment*, es mira si les fams dels animals que mengen el vegetal en qüestió supera el 70%, retornarà un valor més petit per a que surti l'acció de plantar per després donar-li a l'animal.

- Recollir fusta:

  Aquesta acció afecta positivament al *GoalStocks*, ja que augmenta la reserva de fusta. També afecta al *GoalConstruir* de forma negativa, ja que per poder construir una edificació es necessita invertir 3 unitats de fusta.

  - Acció Recollir Fusta (*GoalStocks* - 2, *GoalConstruir* + 1)

  Pel *GoalStocks*, aquesta acció calcula el valor de *discontentment* de la mateixa manera que les accions de plantar vegetals. Si que canvia en el cas del *GoalConstruir*: si falten edificacions per construir, retornarà un nombre més petit que si ja es troben totes en bon estat, per incentivar a recollir fusta per dur a terme aquestes construccions.

5.2.3 Accions: Funcions contínues

En aquest cas, el valor per quantificar com una acció afectarà a un objectiu (*discontentment*) ve donat per una funció contínua [8]. A continuació es detalla per a cada acció quina funció contínua s'ha fet servir per valorar els objectius.

Els valors de *discontentment* estan normalitzats entre 0 i 1. Si una acció no afecta a un objectiu, no té cap funció contínua assignada i retorna directament un valor de 0, per a que no afecti ni positiva ni negativament al valor de *discontentment* final. La manera en que cada acció afecta a un objectiu és igual que en la versió de valors discrets descrita a la secció anterior.

- Accions Construir pont i Construir tanca:

  Les accions de construir repercuteixen sobre dos objectius: *GoalConstruir* i *GoalStocks*. Pel primer objectiu, es mira com està l'edificació en qüestió. Si es troba en bon estat no té sentit dur a terme l'acció, pel que retornarà el valor màxim de *discontentment*. Però si està en runes retorna el valor mínim per a que l'acció tingui possibilitats de ser escollida.

  Aquest valor ve donat per la següent fórmula, sent x la variable d'estat de l'edificació. S'ha de recordar que l'estat de la tanca i del pont serà 1 quan es trobin en bon estat, i 0 quan es trobin en runes.

  $$d(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

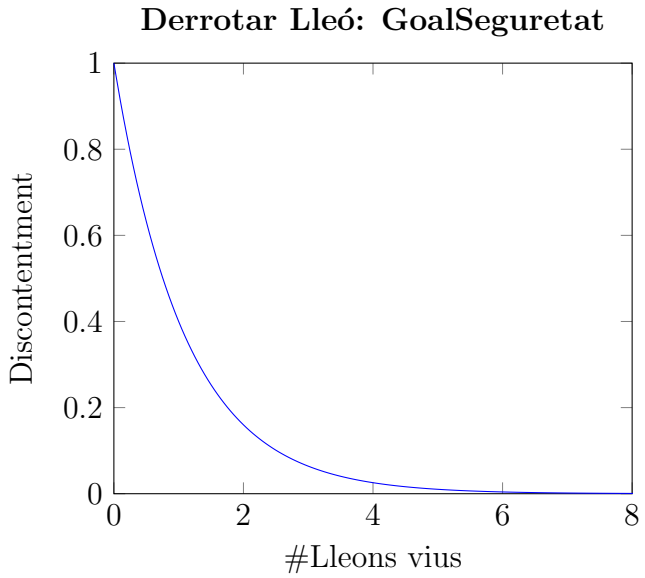
  Sobre el segon objectiu afecta per què cada vegada que es construeix s'han de gastar 3 unitats de fusta. Per tant, si es tenen menys de 4 unitats retornarà el valor màxim de *discontentment*, ja que o bé no seria possible dur a terme la construcció, o bé només quedaria 1 unitat de fusta disponible, i això no interessa a aquest objectiu degut a que la seva finalitat és tenir el màxim nombre de reserves disponibles. Aquest valor s'obté de la següent fórmula, sent x el nombre d'unitats de fusta que es tenen.

  $$d(x) = \begin{cases} 1 & x \leq 3 \\ 0.5 & x > 3 \end{cases}$$

- Acció Derrotar a un lleó:

  Tal i com es mostra en el següent gràfic, aquesta acció afecta al *GoalSeguretat*. El valor de *discontentment* de l'acció Derrotar a un lleó ve donada directament pel nombre de lleons vius que hi ha (sent aquest el valor de x a la fórmula). En el cas extrem que tots els lleons estiguin vius (8) retorna el mínim valor de *discontentment* (0), pel contrari, si estan tots morts, retorna el màxim valor. (1).

  $$d(x) = 0.4^x$$



- Acció Derrotar al capità pirata:

  Aquesta acció afecta només al *GoalSeguretat*, i per valorar-la es té en compte l'estat del capità pirata. Si ja se l'ha derrotat, retorna un valor màxim pel *discontentment* ja que no té sentit realitzar-la, mentre que si està viu retorna el mínim per a que tingui moltes possibilitats de ser escollida. S'ha de recordar que l'estat del pirata és 1 quan aquest es troba derrotat, i 0 quan està viu. Aquest valor obtingut es representa amb la següent fórmula, sent x l'estat actual del capità.

$$d(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

- Accions Alimentar el pollastre, Alimentar la vaca i Alimentar el conill:

  Les accions d'alimentar als animals de granja repercuteixen sobre 2 objectius: *GoalAlimentar* i *GoalStocks*.

  Pel *GoalAlimentar* es tenen en compte dues variables a l'hora de calcular el seu *discontentment*: que es tinguin unitats del vegetal necessari per alimentar l'animal (si no seria impossible de dur a terme l'acció) i la gana del propi animal.

  Per la part de saber si es tenen stocks del vegetal necessari, la funció que s'aplica és la següent, sent x el nombre d'unitats del vegetal que es tenen. Per fer el càlcul del valor de *discontentment* final, a aquest factor se li dona un pes del 20%.

$$factorStock(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

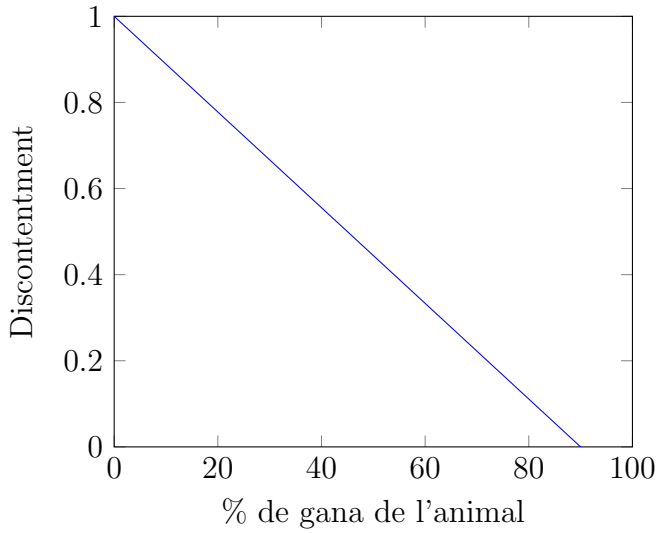
  Per la part de veure com es troba l'animal de famolenc, s'aplica la següent fórmula, sent x la gana de l'animal en qüestió. Com aquesta segona variable és la que més influeix al càlcul del valor de *discontentment* final, se li ha donat un pes major, concretament del 80%.

$$factorGana(x) = -\frac{x}{90} + 1$$

El valor de *discontentment* final per aquest objectiu surt d'aplicar la següent fórmula:

$$d(factorGana, factorStocks) = 0.8 \times factorGana + 0.2 \times factorStocks$$

Alimentar pollastre, vaca, conill: GoalAlimentar



Pel *GoalStocks*, si es tenen més de 15 unitats del vegetal, retorna un valor petit (0.5), per indicar que es pot "gastar"en alimentar l'animal. Si es tenen menys d'aquest nombre d'unitats, retorna el valor màxim per a que l'acció no tingui tantes possibilitats de ser escollida, ja que per aquest objectiu el més important és tenir les màximes reserves de cada aliment i material. La fórmula que s'aplica és la següent, sent x el nombre d'unitats del vegetal corresponent.

$$d(x) = \begin{cases} 1 & x \leq 15 \\ 0.5 & x > 15 \end{cases}$$

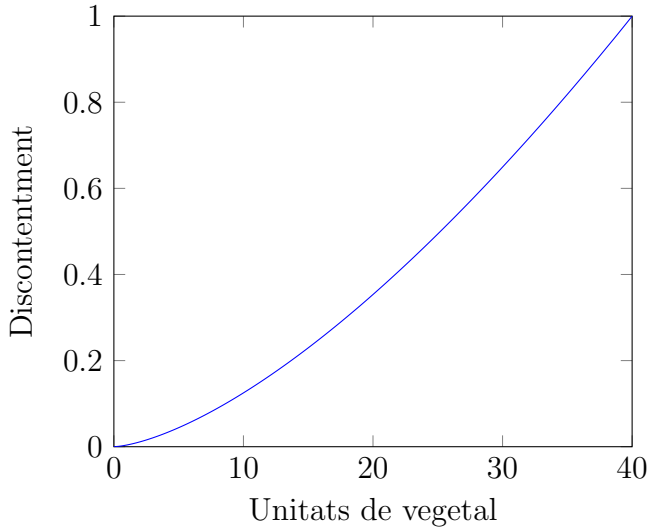
- Accions Plantar herba i Plantar pastanaga:

  Les accions de plantar vegetals afecten a dos objectius: *GoalStocks* i *GoalAlimentar*.

  GoalStocks és el més important per a valorar aquesta acció. És té en compte el nombre d'unitats que es tenen guardades en el magatzem del vegetal corresponent, i el valor de *discontentment* ve donat per la següent funció, sent x el nombre d'unitats del vegetal:

$$d(x) = \frac{1}{40^{1.5}} \times x^{1.5}$$

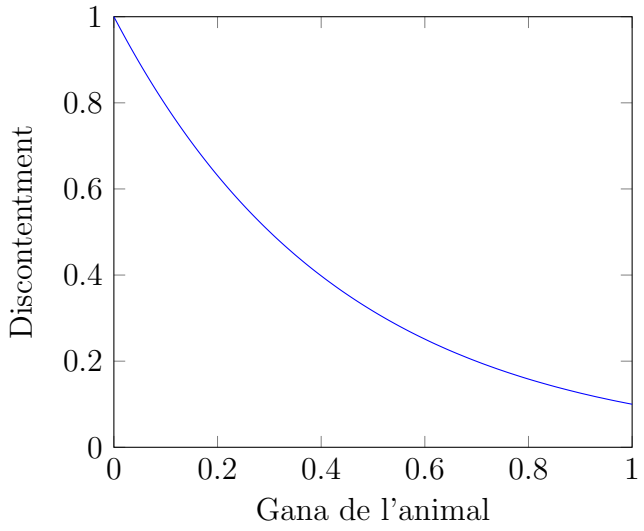
Plantar herba, pastanaga: GoalStocks



Pel segon objectiu al que afecta, el *discontentment* ve donat a partir de la gana que té l'animal que menja el vegetal en qüestió (normalitzant el percentatge), i s'obté el seu valor a partir de la següent funció, sent x aquest valor de gana. Tal com es mostra també en la gràfica següent, mai podrà arribar a retornar un valor de 0 (el mínim) ja que el factor realment decisiu per escollir aquesta acció és el nombre d'unitats que es tenen del vegetal, no pas la gana dels animals.

$$d(x) = 0.1^x$$

Plantar herba, pastanaga: GoalAlimentar



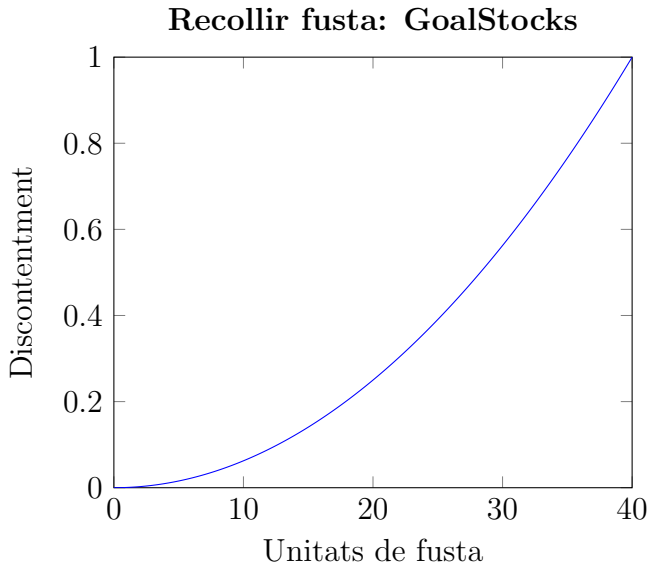
- Acció Recollir fusta:

  De la mateixa manera que les accions de plantar vegetals, l'acció de recollir fusta afecta a 2 objectius: *GoalStocks* i *GoalConstruir*.

  Pel primer, es té en compte el nombre d'unitats de fusta que es tenen guardades en el magatzem, i el valor de *discontentment* resultant ve donat per la

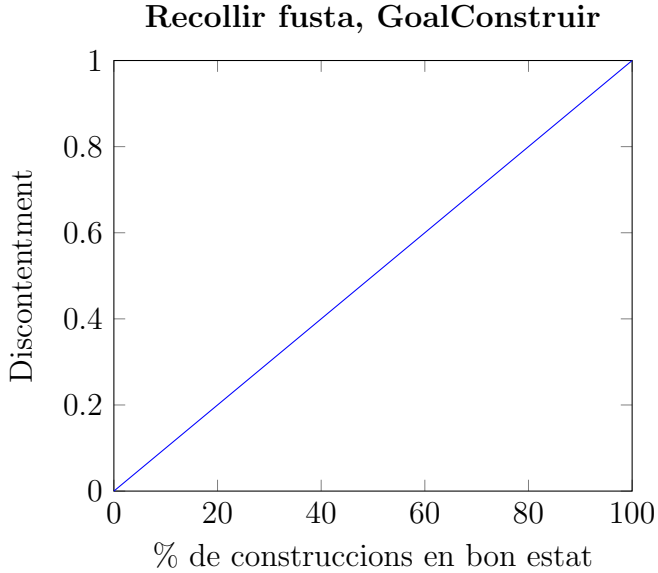
següent funció, sent x el nombre d'unitats de fusta.

$$d(x) = \frac{1}{40^2} \times x^2$$



Pel segon objectiu al que afecta, el *discontentment* ve donat directament pel percentatge d'edificacions que es troben en bon estat, i s'obté el seu valor a partir de la següent funció, sent x aquest percentatatge d'edificacions en bon estat:

$$d(x) = \frac{x}{100}$$



Un cop definides les funcions i valors que contemplarà la IA per trobar els *discontentments* de cada acció, es detallen els diagrames de seqüència per veure el funcionament principal de l'algorisme.

5.2.4 Diagrames de Seqüència

El diagrama de seqüència del cas d'ús que s'executa quan el jugador interactua amb en Lord Barus utilitzant GOB sense tenir en compte el temps es pot veure a la Figura 7.

El cas d'ús comença quan el jugador interactua amb en Lord Barus per a que li assigni una missió. En aquell moment es dispara un esdeveniment, capturat per la classe `QuestNPCClickEvent`, i es va cridant a un seguit de classes ja implementades pel joc: `MissionsSelectorMenu`, la classe que s'encarrega de mostrar la informació de les missions en la UI, `QuestManager`, utilitzada per llançar esdeveniments de missions i monitoritzar el seu estat, fins arribar a `QuestDetailArchive`. Aquesta és la classe encarregada d'emmagatzemar tota la informació de les missions per a retornar la que toqui al jugador. És la que crida al mètode de la classe `ActionChooser` per executar l'algorisme GOB.

Per a cada acció, comprova quin *discontentment* produeix a cada objectiu, que ve donat per dos valors: com afecta l'acció al objectiu (a la Figura 7 correspon a la crida al mètode `getGoalChange()`), i el valor propi de l'objectiu en aquell instant determinat (a la Figura 7 correspon a la crida al mètode `getValue()`). Es queda amb l'acció que li retorni un valor de *discontentment* més baix.

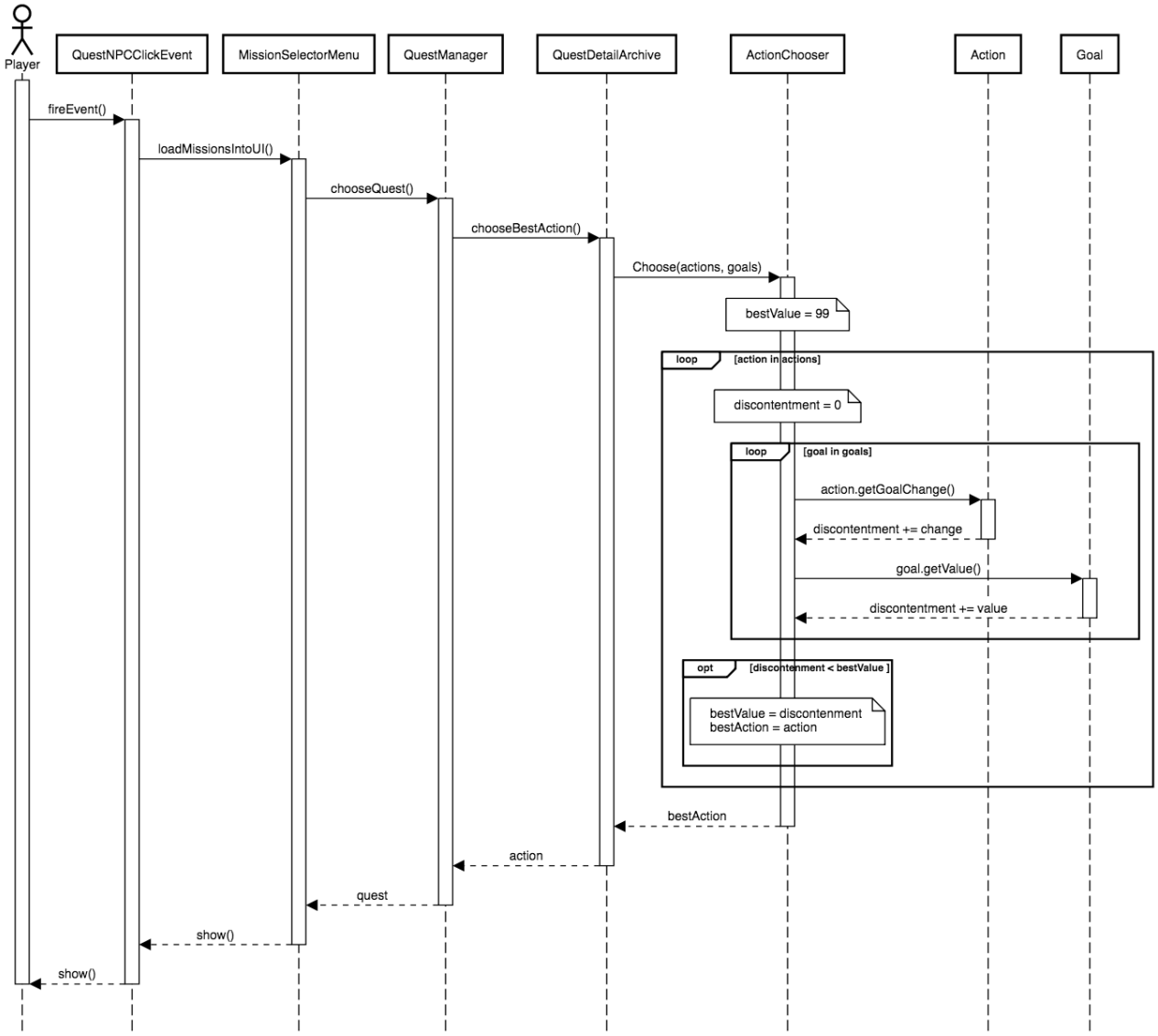


Figura 7: Diagrama de Seqüència GOB sense temps.

En l'aproximació on es té en compte el temps, el diagrama de seqüència es veu modificat en la part de la classe `ActionChooser`, on en el mètode de seleccionar l'acció més adient, el *discontentment* també es veu afectat pel valor de com canvia l'objectiu en el temps que triga en dur-se a terme l'acció. Es pot veure en la Figura 8, concretament en la crida al mètode `getDuration` de l'acció en qüestió.

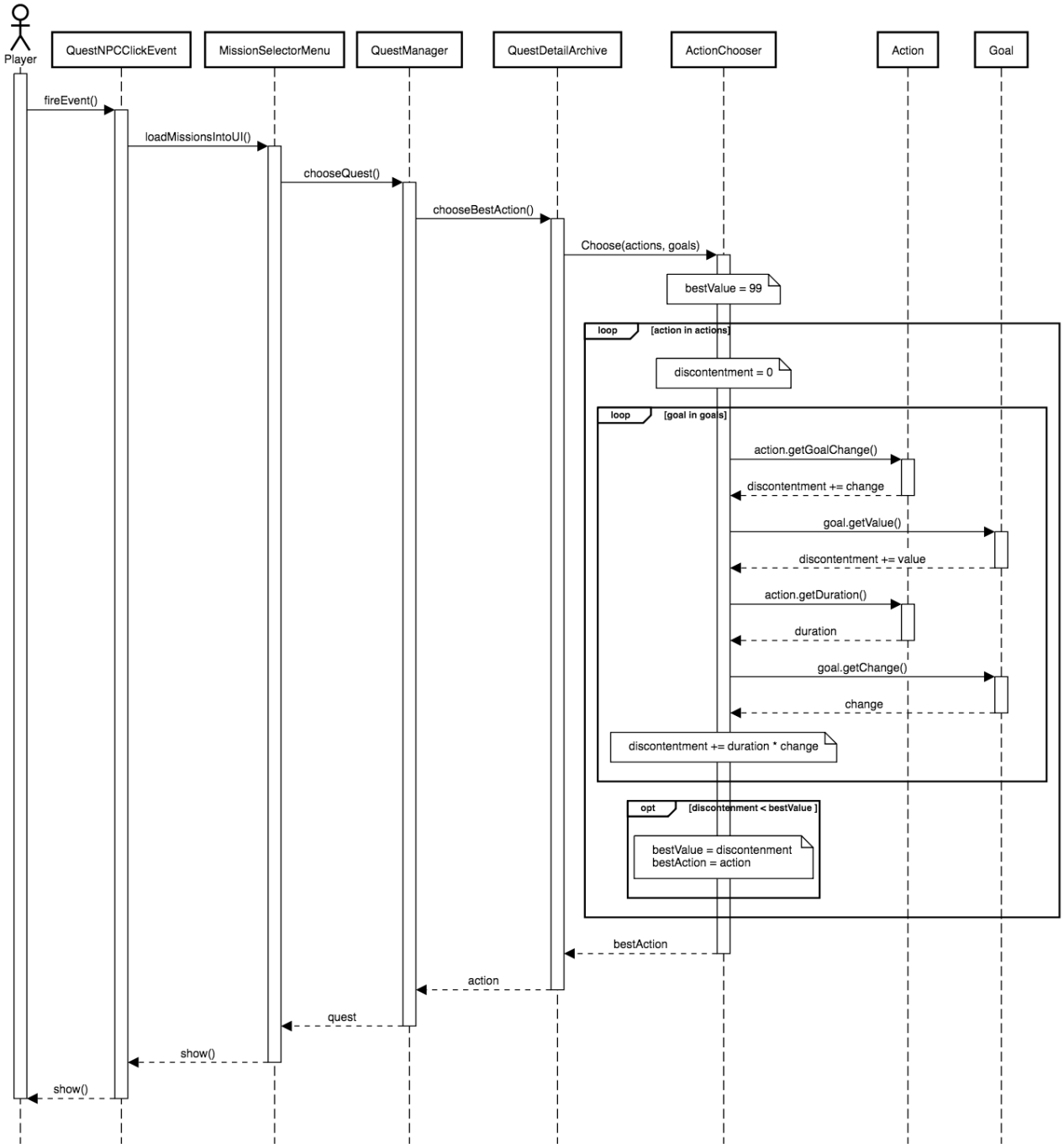


Figura 8: Diagrama de seqüència GOB amb temps.

5.3 Aproximació 2: Utility system

El sistema basat en utilitats [9] també assigna un valor d'utilitat a cada acció depenent de l'estat de les variables del món, però sense tenir en compte com es troben els objectius.

En aquest cas es calcula el valor d'utilitat per a cada acció i el sistema es queda amb la que té un valor més alt, indicant que serà l'acció que afecti de manera més

positiva al món.

Al no tenir en compte com es troben els objectius, no té sentit aplicar el càlcul del temps tal com s'ha plantejat en la implementació de GOB en el Sistema d'Utilitats.

No és un món simple, per tant és raonable que les accions depenguin de més d'una variable d'estat del món. Per exemple, per l'acció *Alimentar al conill*, s'haurà de tenir en compte la gana que té el conill i si es tenen pastanagues. Per això s'introdueixen els factors de decisions. Afegint pesos a cada variable, es pot determinar i quantificar quina és més important a l'hora d'obtenir una utilitat per l'acció pertinent.

Tot i així, només amb aquestes variables prèviament definides (veure Secció 5.1), s'obtenien uns factors de decisió molt simples, per tant per aquesta tècnica s'han definit encara més variables que podrien acabar repercutint sobre les accions que pot realitzar el NPC. Es pot veure com s'enllacen en les variables d'estat del joc i les accions que hi han en la Figura 9.

Les noves variables introduïdes son les següents:

- Water amount: quantitat d'aigua que es té. Funciona com els stocks d'aliments i materials. S'utilitzaria a l'hora de conrear un vegetal per regar-lo i fer-lo créixer.
- Milk amount: quantitat de llet que es té. Funciona com els stocks d'aliments i materials. Quan una vaca estigués al 100% satisfeta de gana, podria donar llet per al consum de la població.
- Player health: Percentatge de vida que té el jugador.
- Dead population: Número de vilatans que han mort (a causa dels lleons o del capità pirata).
- Total population: Número total de població (constant).

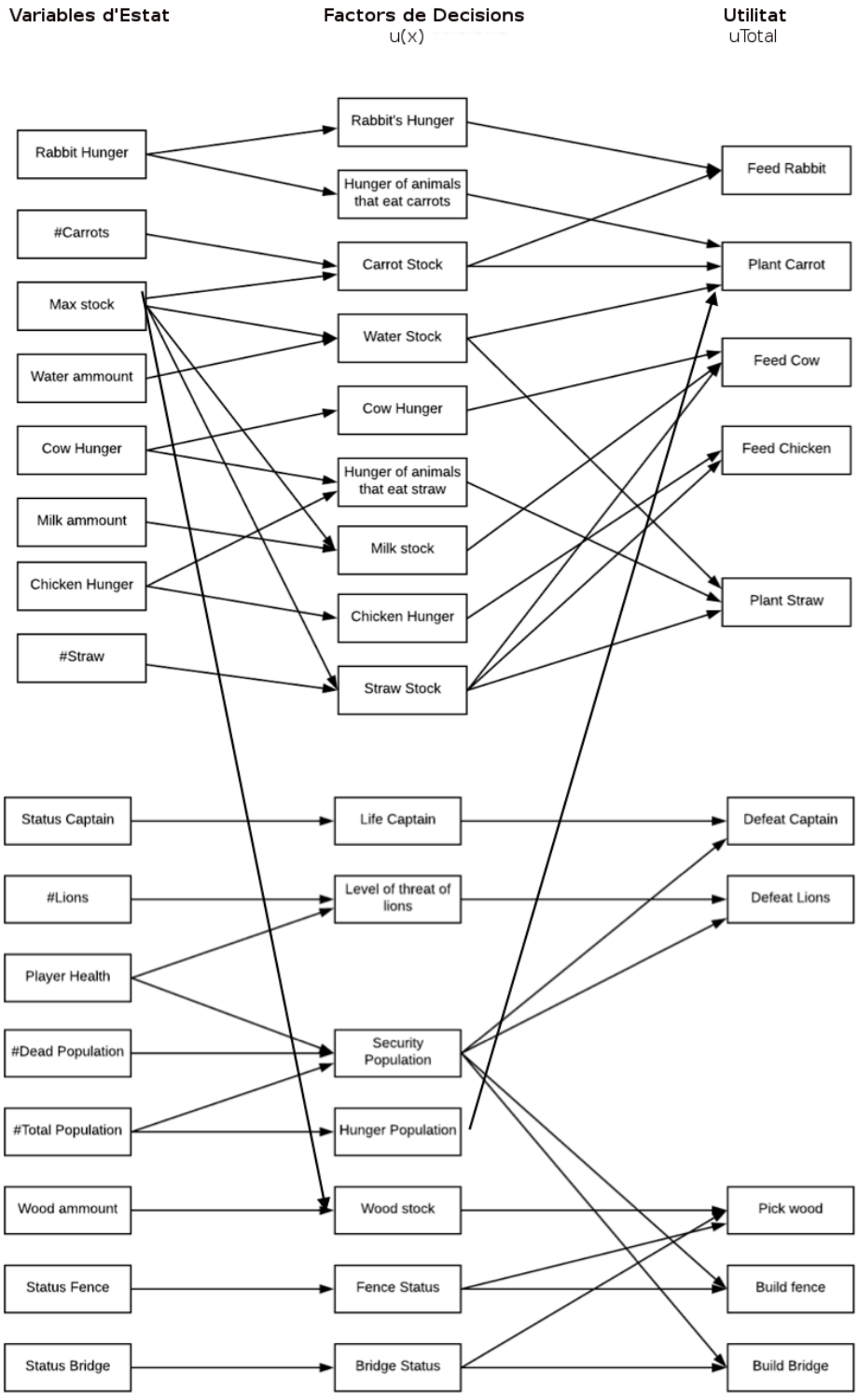


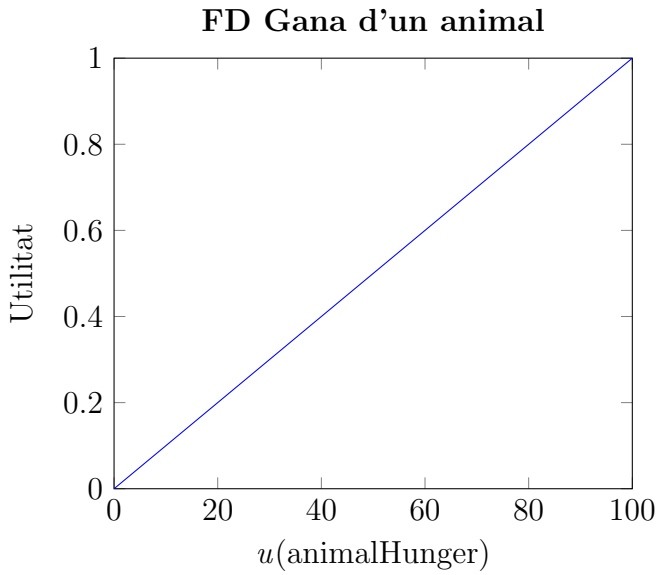
Figura 9: Factors de decisions a Fracsland. La primera columna correspon a les variables del món, la segona als factors de decisions propis i la tercera a les accions. Més endavant es detallen els càlculs dels factors de decisions, referenciats amb *u(x)*, i el càlcul de les utilitats finals de les accions, referenciats com *uTotal*.

Els factors de decisions (FD) s'han pensat de la següent manera:

- Rabbit's hunger: percentatge, es calcula a partir de la mitjana de la gana de tots els conills (encara que en el joc actual només n'hi ha un).
- Cow's hunger: percentatge, es calcula a partir de la mitjana de la gana de totes les vaques (encara que en el joc actual només n'hi ha una).
- Chicken's hunger: percentatge, es calcula a partir de la mitjana de la gana de tots els pollastres (encara que en el joc actual només n'hi ha un).
- Hunger of animals that eat carrots: percentatge, es calcula fent la mitjana de la gana de tots els animals que menjen pastanaga (en el joc actual, només el conill).
- Hunger of animals that eat straw: percentatge, es calcula fent la mitjana de la gana de tots els animals que mengen herba (en el joc actual, la vaca i el pollastre).

  Els factors de decisions esmentats fins ara (els relacionats amb la gana dels animals), calculen la seva utilitat a partir de la següent funció, on x és la gana de l'animal o mitjana de gana dels animals en qüestió:

$$u(x) = \frac{x}{100}$$

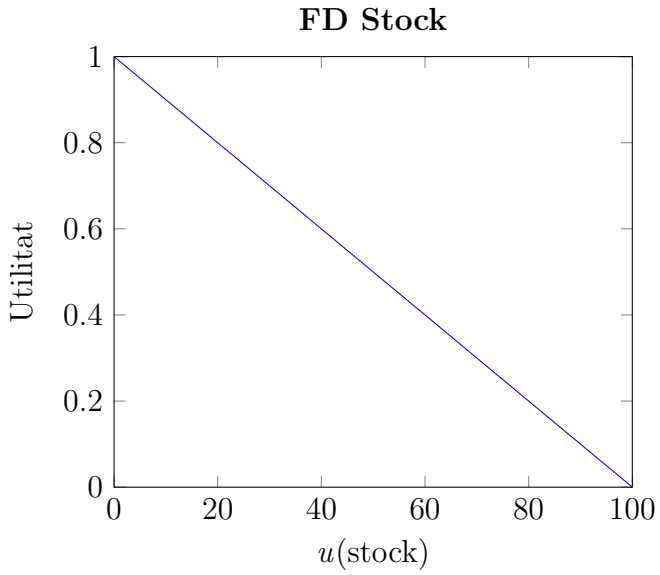


- Water Stock, Milk Stock, Wood Stock, Straw Stock, Carrot Stock: Unitats que es tenen del aliment o material dividit entre el màxim que es pot guardar (o dit d'una altra manera, com de ple es troba el magatzem per a aquell aliment o material).

  Aquests factors de decisions relacionats amb les reserves d'aliments i materials calculen la seva utilitat aplicant la següent fórmula, sent x el valor de stock

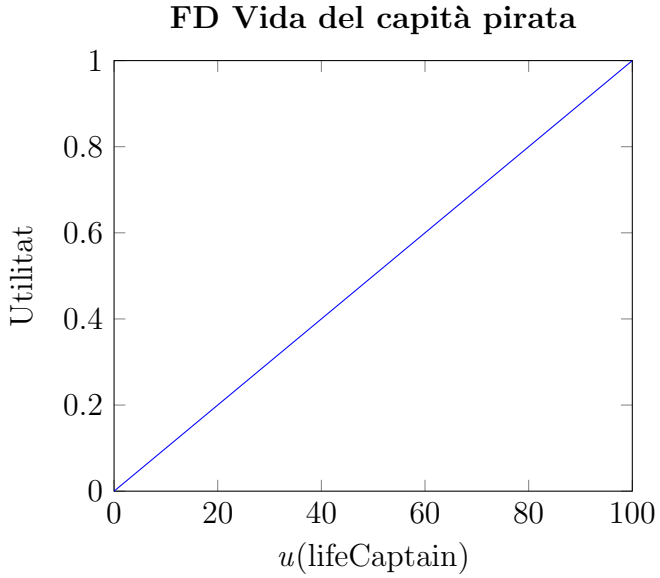
del material o vegetal en qüestió:

$$u(x) = 1 - \frac{x}{100}$$



- Life Captain: Percentatge. és el nivell de vida que té el capità. Quan més nivell de vida tingui més difícil serà derrotar-lo. Es calcula la seva utilitat de la següent manera, sent x la vida del capità:

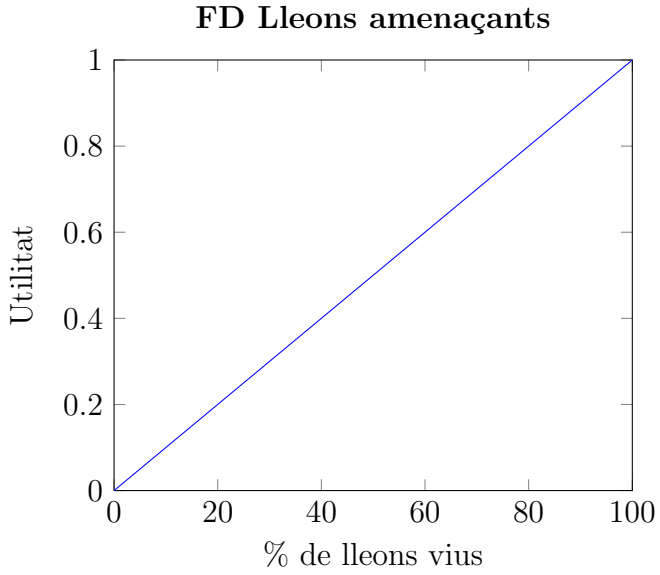
$$u(x) = \frac{x}{100}$$



- Level of threat of lions: es calcula a partir del nombre de lleons vius i del nivell de vida del jugador.

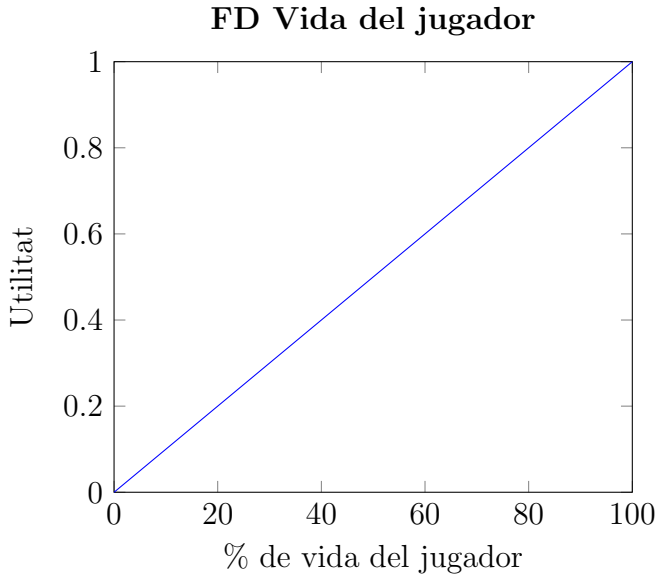
Pel factor de lleons vius restants, quant menys lleons vius quedin, menys amenaçaran a la població. Això es pot quantificar de la següent manera:

$$factorLleons = \frac{numLleons}{totalLleons}$$



La vida del jugador influeix de la forma que si el jugador té molta vida serà més fàcil que pugui derrotar als lleons.

$$factorJugador = \frac{nivellVida}{100}$$

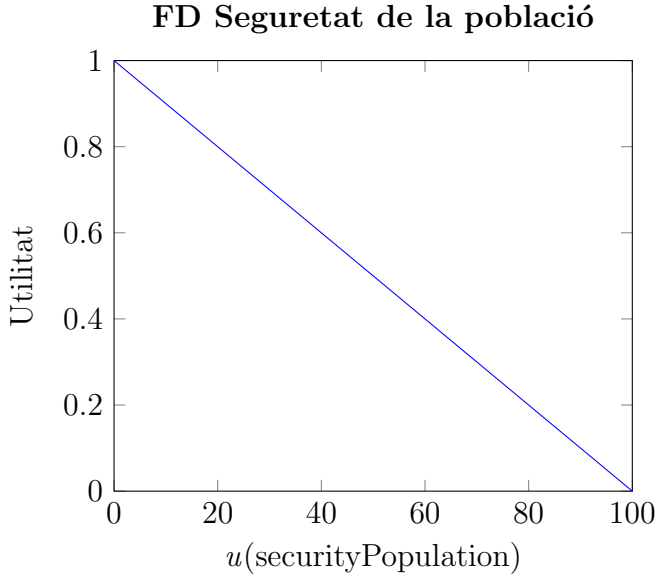


La utilitat final, *u(threatLions)*, es calcula d'aquesta manera:

$$u(factorJugador, factorLleons) = 0.3 \times factorJugador + 0.7 \times factorLleons$$

- Security population: és el nivell de seguretat de la població. Quanta més població viva hi hagi, més seguretat hi haurà. La seva utilitat es calcula aplicant la següent fórmula, sent x el nombre d'habitants vius que hi ha:

$$u(x) = 1 - \frac{x}{totalHabitants}$$



- Hunger population: Correspon al percentatge de gana que té la població. Es calcula a partir de la població viva que hi ha. Per no complicar-ho, a cada vilatà li augmentaria la gana de la mateixa manera i s'alimentarien tots junts, així que la utilitat es calcularia utilitzant la mateixa fórmula que per la gana dels animals de granja.

- Fence status: booleà, indica si la tanca està construïda o no.

- Bridge status: booleà, indica si el pont està construït o no.

  Per aquests dos factors de decisions relacionat amb l'estat de les edificacions, el valor de la seva utilitat es calcula de la següent manera, sent x l'estat del pont o de la tanca, és a dir *u(statusBuilding)*:

$$u(x) = \begin{cases} 1 & x \leq 0 \\ 0 & x > 0 \end{cases}$$

Un cop definits els factors de decisions, es defineix a continuació com afecten a cada acció. En el cas del món simple (l'actual de Fracsland) serien molt semblants, només canviarien en que ometen les variables noves introduïdes en els càlculs.

- Construir pont, Construir tanca:

  Per calcular el valor de la utilitat d'aquestes accions, en el cas que l'edificació en qüestió estigui sense construir, es tenen en compte amb la mateixa importància el stock de fusta i la seguretat de la població (ja que les edificacions poden servir per protegir a la població dels lleons i pirates).

  $$uTotal = 0.5 \times u(woodStock) + 0.5 \times u(securityPopulation)$$

  Si l'edificació ja es troba construïda, retorna la mínima utilitat per a que no s'esculli.

- Derrotar a un lleó:

  Per calcular el valor de la utilitat de l'acció Derrotar a un lleó es té en compte el nivell de seguretat de la població i el nivell d'amenaça dels lleons. S'utilitza la següent fórmula:

  $$uTotal = 0.6 \times u(securityPopulation) + 0.4 \times u(threatLions)$$

- Derrotar al capità pirata:

  Per calcular el valor de la utilitat de l'acció Derrotar al capità pirata es té en compte la vida del capità i el nivell de seguretat de la població, i s'utilitza la següent fórmula:

  $$uTotal = 0.5 \times u(lifeCaptain) + 0.5 \times u(securityPopulation)$$

- Alimentar el pollastre, Alimentar el conill:

  En aquest cas les accions d'alimentar als 3 animals de granja no utilitzen la mateixa fórmula per calcular la seva utilitat. Pel que fa aquestes dues accions, es té en compte la gana de l'animal i quant de stock es té del vegetal que necessita per alimentar-se. S'utilitza la següent fórmula:

  $$uTotal = 0.7 \times u(animalHunger) + 0.3 \times u(vegetableStock)$$

- Alimentar la vaca:

  Aquesta acció varia de les dues anteriors ja que quan una vaca no està famolenca produeix una unitat de llet. Així a part de la gana de l'animal i el stock de herba, també s'ha de tenir en compte el stock de llet. Si es té poca llet, convendrà fer aquesta acció per produir-ne. S'utilitza la següent fórmula:

  $$uTotal = 0.4 \times u(cowHunger) + 0.4 \times u(milkStock) + 0.2 \times u(strawStock)$$

- Plantar herba:

  Per calcular la utilitat d'aquesta acció es té en compte la gana dels animals que mengen herba, el seu stock actual i el stock d'aigua, ja que per a conrear-la s'ha de regar l'hort. S'utilitza la següent fórmula:

$$uTotal = 0.4 \times u(hungerAnimalsThatEatStraw) + 0.4 \times u(strawStock)$$
$$+0.2 \times u(waterStock)$$

- Plantar pastanaga:

  Per calcular la utilitat d'aquesta acció es fa de la mateixa manera que l'anterior però afegint la gana de la població, ja que s'alimenten amb pastanagues. S'utilitza la següent fórmula:

$$uTotal = 0.4 \times u(hungerAnimalsThatEatCarrots) + 0.4 \times u(carrotStock)$$
$$+0.1 \times u(waterStock) + 0.1 \times u(hungerPopulation)$$

- Recollir fusta:

  Per calcular la utilitat d'aquesta acció es té en compte el stock actual de fusta i l'estat de les construccions, ja que per construir-les es necessita fusta. S'utilitza la següent fórmula:

$$uTotal = 0.3 \times (0.5 \times u(statusFence) + 0.5 \times u(statusBridge))$$
$$+0.7 \times u(woodStock)$$

5.3.1 Diagrama de Seqüència

Amb aquesta aproximació, el diagrama de seqüència és pràcticament igual que al de GOB fins arribar a la classe `ActionChooser`. Com s'ha explicat, no es tenen en compte els objectius, per tant només itera sobre les accions per retornar la que tingui un valor d'utilitat més gran. Es pot veure a la Figura 10.

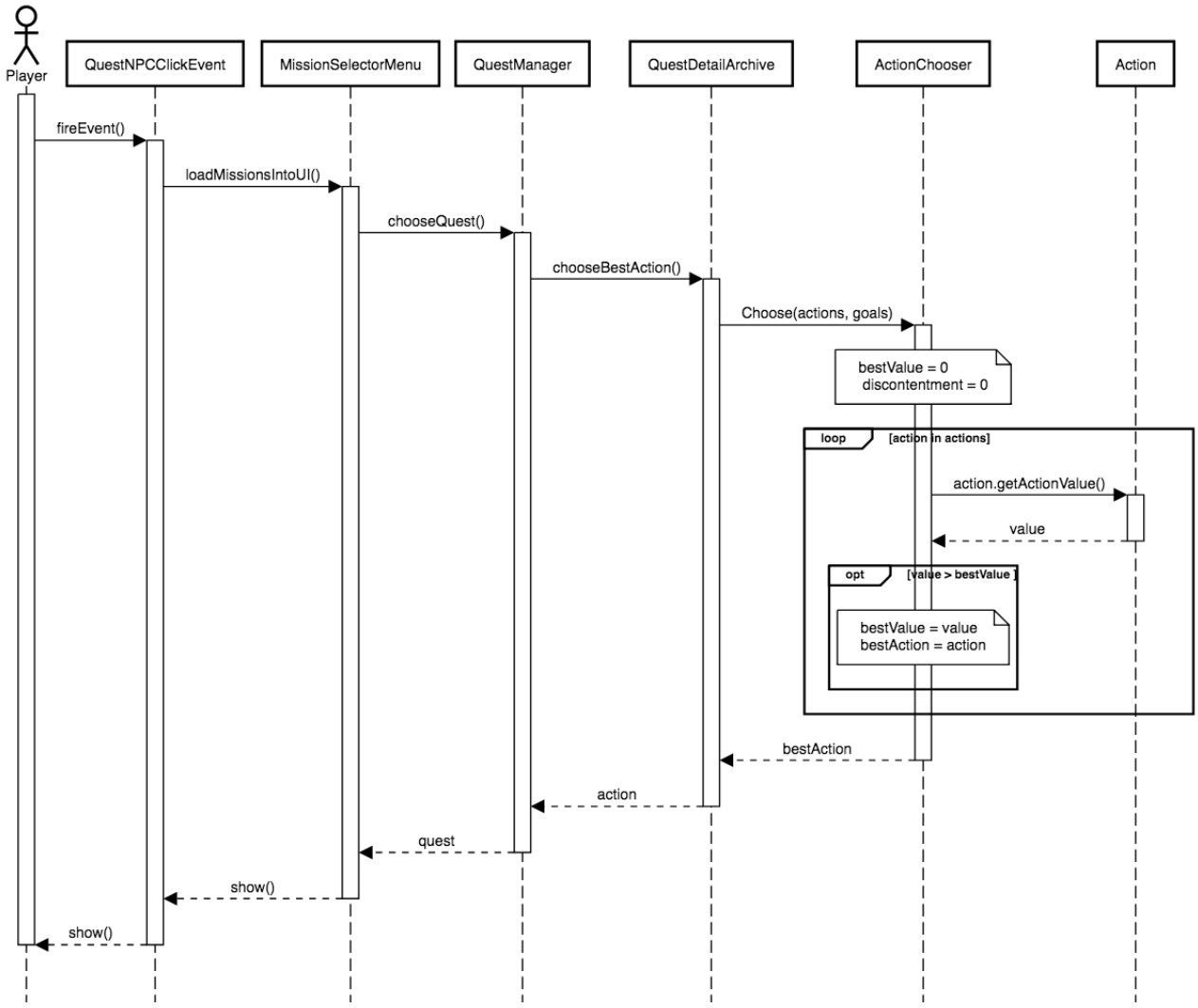


Figura 10: Diagrama de seqüència Utility System.

6 Desenvolupament del joc

Tenint en compte que aquest projecte és una extensió d'un treball fet anteriorment, utilitzar Unity era la millor opció ja que és com originalment es va implementar Fracsland.

Actualment ja existeixen puglins de Unity que permeten implementar les funcionalitats que es volen aconseguir amb aquest projecte. Un exemple és DecisionFlex [10], que implementa una Intel·ligència Artificial basada en el Sistema d'Utilitats, però per la realització d'aquest treball s'ha considerat més adequat implementar la lògica de les diferents tècniques d'IA des de zero i completament enfocades a Fracsland. D'aquesta manera és més fàcil fer l'estudi i també és fa més senzill poder comparar les diferents tècniques implementades, ja que si s'utilitzessin plugins diferents per a cada una, seria més difícil de trobar punts en comú a causa de les diferències en les implementacions.

Així mateix, implementar els algorismes de presa de decisions en C# no suposa cap problema.

6.1 Modificació del joc inicial Fracsland

Inicialment Fracsland estava pensat per ser jugat amb un servidor remot. Aquest implementava tota la part de persistència de dades del jugador i interacció amb el joc per personalitzar les partides, però aquest és un aspecte irrellevant per aquest projecte. El codi també estava preparat per a poder jugar utilitzant un servidor local, en principi per a fer testeig "*fake*", així que es va implementar un petit servidor en Java per permetre un flux bàsic de comunicació entre el client i el servidor per enviar i rebre les peticions corresponents.

Un cop solucionada la part que involucrava el servidor, es van trobar petits *bugs* i millores durant l'execució del joc que també es van arreglar:

- Bug: Donava error a l'intentar plantar un vegetal a l'hort sense estar en una missió. Això ha de ser possible ja que és un joc de món obert on el jugador es pot moure lliurement independentment de la missió on es trobi.
- Millora: Al talar una palmera, el personatge quedava inclinat en diagonal mirant cap al cel.
- Ampliació: Amagar/mostrar el minimap que apareix a la dreta de la pantalla en qualsevol moment de la partida. S'ha afegit aquesta petita funcionalitat perquè depenent del tamany de la pantalla on es jugui Fracsland, el minimapa ocupa bastant espai respecte la càmera principal del joc.
- Ampliació: Poder fer zoom in o zoom out en qualsevol moment de la partida amb la rodeta del ratolí.

Afegir aquests canvis també va servir per familiaritzar-se amb el codi de Fracsland.

7 Resultats i Simulacions

Aquesta secció detalla totes les simulacions i estudis que s'han fet sobre les tècniques i les seves variacions explicades al Capítol 5 (Disseny de la IA), així com les conclusions a les que s'han arribat després de comparar-les.

7.1 Simulacions

Per fer les simulacions s'han executat els diferents algorismes amb diferents números d'iteracions simulant que es duia a terme l'acció seleccionada a cada iteració. S'han fet taules comparatives per poder diferenciar les accions resultants escollides segons la tècnica utilitzada i com queda l'estat del joc a llarg termini.

Per a tots els casos s'ha definit una nomenclatura per anomenar a cada versió dels algorismes implementats:

- *GOB_DIS*: GOB utilitzant valors discrets.
- *GOB_DIS_T*: GOB utilitzant valors discrets i tenint en compte el temps que triga cada acció.
- *GOB_CONT*: GOB utilitzant funcions contínues.
- *GOB_CONT_T*: GOB utilitzant funcions contínues i tenint en compte el temps que triga cada acció.
- *GOB_CONT_C*: GOB utilitzant funcions contínues i amb el món complex (tenint en compte les noves variables afegides).
- *UTLT*: Sistema d'utilitats.
- *UTLT_C*: Sistema d'utilitats amb el món complex (tenint en compte les noves variables afegides).

7.1.1 Comparació d'accions en un nombre d'iteracions determinat

Per fer aquest estudi s'ha executat cada algorisme fent 30 iteracions per veure quina acció escollia a cada moment i l'estat final del joc. Els resultats es poden veure a la Taula 1.

S'ha de tenir en compte que no només canvia el món quan es realitza l'acció que surt, sinó que també succeeixen coses de forma automàtica: a les iteracions 4, 16 i 25 neix un nou lleó, el pont es destrueix automàticament a les iteracions 14 i 23, la tanca ho fa a les iteracions 10 i 21 i el capità pirata es recupera a les iteracions 17 i 27.

	GOB_DIS	**GOB_DIS_T**	**GOB_CONT**	**GOB_CONT_T**	**UTLT**	**GOB_CONT_C**	**UTLT_C**
1	Defeat Pirate	Defeat Pirate	Defeat Pirate	Pick Wood	Plant Straw	Pick Wood	Plant Straw
2	Plant Straw	Plant Straw	Plant Straw	Plant Straw	Defeat Lion	Defeat Pirate	Defeat Lion
3	Plant Carrot	Plant Carrot	Plant Carrot	Plant Carrot	Plant Carrot	Plant Straw	Pick Wood
4	Pick Wood	Defeat Lion	Plant Straw	Plant Straw	Pick Wood	Plant Carrot	Defeat Pirate
5	Build Bridge	Pick Wood	Plant Carrot	Plant Carrot	Defeat Pirate	Plant Straw	Plant Straw
6	Pick Wood	Build Bridge	Plant Straw	Pick Wood	Plant Straw	Plant Carrot	Pick Wood
7	Pick Wood	Pick Wood	Plant Carrot	Plant Straw	Plant Carrot	Pick Wood	Plant Straw
8	Pick Wood	Pick Wood	Defeat Lion	Plant Carrot	Pick Wood	Build Bridge	Build Bridge
9	Pick Wood	Pick Wood	Defeat Lion	Defeat Lion	Plant Straw	Plant Straw	Plant Carrot
10	Build Fence	Pick Wood	Defeat Lion	Defeat Lion	Plant Carrot	Plant Carrot	Plant Carrot
11	Plant Straw	Build Fence	Defeat Lion	Defeat Lion	Plant Straw	Feed Cow	Feed Cow
12	Plant Carrot	Plant Carrot	Plant Straw	Defeat Lion	Feed Cow	Defeat Lion	Plant Straw
13	Feed Chicken	Feed Chicken	Plant Carrot	Build Fence	Feed Chicken	Defeat Lion	Plant Carrot
14	Feed Rabbit	Feed Rabbit	Defeat Lion	Build Bridge	Feed Rabbit	Defeat Lion	Plant Carrot
15	Feed Cow	Feed Cow	Pick Wood	Build Bridge	Plant Carrot	Build Bridge	Build Bridge
16	Build Bridge	Build Bridge	Build Fence	Defeat Pirate	Plant Straw	Feed Chicken	Feed Rabbit
17	Build Fence	Build Fence	Build Bridge	Plant Straw	Build Fence	Feed Rabbit	Feed Chicken
18	Plant Straw	Defeat Lion	Pick Wood	Defeat Lion	Defeat Lion	Plant Straw	Defeat Lion
19	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate
20	Plant Carrot	Pick Wood	Pick Wood	Plant Carrot	Pick Wood	Feed Cow	Plant Straw
21	Plant Straw	Pick Wood	Feed Cow	Feed Cow	Build Bridge	Defeat Lion	Feed Cow
22	Plant Carrot	Pick Wood	Feed Rabbit	Feed Rabbit	Feed Cow	Plant Carrot	Plant Straw
23	Build Fence	Build Fence	Build Fence	Build Fence	Build Fence	Build Fence	Plant Straw
24	Defeat Lion	Pick Wood	Feed Chicken	Feed Chicken	Feed Chicken	Plant Straw	Feed Cow
25	Feed Chicken	Feed Chicken	Build Bridge	Build Bridge	Build Bridge	Build Bridge	Pick Wood
26	Feed Rabbit	Feed Rabbit	Plant Straw	Plant Straw	Feed Rabbit	Feed Cow	Build Bridge
27	Feed Cow	Build Bridge	Defeat Lion	Defeat Lion	Plant Carrot	Defeat Lion	Feed Chicken
28	Build Bridge	Plant Straw	Plant Carrot	Plant Carrot	Plant Straw	Pick Wood	Plant Straw
29	Defeat Pirate	Feed Cow	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate	Defeat Pirate
30	Plant Straw	Defeat Pirate	Pick Wood	Pick Wood	Feed Cow	Feed Chicken	Feed Rabbit

Taula 1: Accions escollides en 30 iteracions de cada versió.

Anàlisis de la taula d'accions per a cada model:

- **Versions de GOB amb valors discrets: *GOB_DIS* i *GOB_DIS_T*:**

  La primera clara diferència que s'aprecia és que al model *GOB_DIS_T* recull molta més fusta que al model *GOB_DIS*. Això és degut a que l'acció *Pick Wood* té una durada molt curta, per tant en la versió que té en compte el càlcul de temps surt més vegades escollida. En tots dos casos és normal que surtin escollides durant les primeres iteracions. Ja que no solament influeix en l'objectiu *GoalStocks*, si no també al *GoalConstruir* (i sense fusta no es poden dur a terme les accions de construir el pont i la tanca).

  Pel mateix motiu, en *GOB_DIS* s'escullen més vegades les accions *Plant Straw* i *Plant Carrot*. Aquestes dues, juntament amb *Pick Wood*, són les que més influencien al *GoalStocks*, però com *Pick Wood* triga menys temps en realitzar-se que *Plant Straw* i *Plant Carrot*, la primera surt escollida més cops en *GOB_DIS_T*, i les altres dues en *GOB_DIS*.

  Una altra similitud que tenen les dues versions de GOB és que les accions *Feed Chicken*, *Feed Cow* i *Feed Rabbit* surtin escollides una rere l'altre, ja que els tres animals comencen sense tenir gana quan s'inicia la partida i van incrementant gradualment la seva gana en la mateixa magnitud a mesura que avança el joc. L'algorisme comença a considerar important l'elecció d'aquestes accions quan la fam de l'animal és superior al 65%, per això és normal que al finalitzar la simulació la gana dels 3 animals quedi per sota d'aquest valor, com es podrà veure a la Taula 2.

  En tots dos models (*GOB_DIS i GOB_DIS_T*) es comença escollint l'acció *Defeat Pirate*. Això és degut a que el nivell d'importància del *GoalSeguretat* és molt alt (s'analitza a la Secció 7.1.2: Comparació de l'estat del joc), i la variable d'estat que més influeix a aquest objectiu és l'estat del capità pirata, que està viu. Per això l'escull com a primera acció, a més és un tipus d'acció que només realitzant-la un cop es satisfà (a diferència per exemple de *Plant Carrot*, que s'ha de fer molts cops fins a que s'ompli el stock). Per aquest mateix motiu, és normal que en tots dos models, al finalitzar la simulació la tanca i el pont es trobin en estat construït: *Built Fence* i *Built Bridge* són accions que només cal fer-les un sol cop per a completar el *GoalConstruir*.

  Una altra diferència és que l'acció *Defeat Lion* surt escollida més cops en el segon model. Això també és degut a que és una acció de curta durada encara que no influeix en gran quantitat al *GoalSeguretat*, pel que si no es té en compte el temps no surt tant a compte realitzar-la com altres accions que sí que influeixen més als seus propis objectius.

- **Versions GOB amb funcions contínues: *GOB_CONT* i *GOB_CONT_T*:**

  En el model *GOB_DIS* la primera acció escollida és *Defeat Pirate*, ja que té un nivell d'importància molt alt, igual que passa amb *GOB_CONT*. En canvi, en el model *GOB_CONT_T*, l'algorisme comença escollint les accions de plantar i recollir stock. Això té sentit perquè *Defeat Pirate* té una de les durades més llargues, mentre que les accions de plantar i recollir són d'una durada

intermèdia o curta i influeixen a varis objectius a la vegada. Per exemple, *Pick Wood* influencia a *GoalStocks* i *GoalConstruir*, i les accions *Plant Straw* i *Plant Carrot* influeixen a *GoalStocks* i *GoalAlimentar*.

Una similitud que tenen tots dos models és que algunes accions surten escollides seguides varis cops, com passa per exemple amb *Defeat Lion*. Això és degut a que el càlcul del *discontentment* es realitza a partir d'una funció contínua, per tant si els valors de les variables que influeixen a les accions varien poc, el valor final de *discontentment* de la funció contínua també serà molt semblant.

Igual que passa a *GOB_DIS*, s'entén que les accions *Feed Chicken*, *Feed Cow* i *Feed Rabbit* surtin escollides juntes, ja que la gana dels 3 animals incrementa en la mateixa quantitat a mesura que avança el joc.

- **Models amb món complex: *GOB_CONT_C* i *UTLT_C*:**

  Tant *GOB_CONT_C* com *UTLT_C* es comporten de forma semblant a les tècniques de món simple *GOB_CONT* i *UTLT* respectivament. Comparades entre elles tampoc es poden extreure unes conclusions diferents que quan es fa córrer l'algorisme amb les versions de món simple amb menys variables d'estats.

7.1.2 Comparació de l'estat del joc

Una altra comparació que s'ha fet ha sigut veure l'estat del joc després d'executar l'algorisme una sèrie d'iteracions. Per això s'han calculat els valors de satisfacció de cada objectiu. Els resultats es poden veure a la Taula 2.

Inicialment els objectius tenen els mateixos valors de satisfacció per a totes les versions. S'han descrit prèviament a la Secció 5.2.1 (Valors dels Objectius), però a continuació es recorden:

- *GoalStocks*: valor inicial de 0, ja que inicialment no es tenen reserves de cap aliment ni material.
- *GoalAlimentar*: valor inicial de 100, ja que al començament els animals no tenen gens de fam.
- *GoalSeguretat*: valor inicial de 0, ja que al principi de la partida tant els lleons com el capità pirata estan vius.
- *GoalConstruir*: valor inicial de 0, ja que a l'inici les edificacions es troben en runes.

Iteració	*GoalStocks*	*GoalAlimentar*	*GoalSeguretat*	*GoalConstruir*
30	GOB_DIS: 13.34% GOB_DIS_T: 15% GOB_CONT: 15.66% GOB_CONT_T: 13.66% UTLT: 13.66% GOB_CONT_C: 10.66% UTLT_C: 13.66%	GOB_DIS: 75% GOB_DIS_T: 78.33% GOB_CONT: 56.67% GOB_CONT_T: 56.66% UTLT: 78.34% GOB_CONT_C: 66.66% UTLT_C: 80%	GOB_DIS: 75.5% GOB_DIS_T: 82.5% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 82.5% GOB_CONT_C: 97.5% UTLT_C: 82.5%	GOB_DIS: 100% GOB_DIS_T: 100% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 100% GOB_CONT_C: 100% UTLT_C: 50%
500	GOB_DIS: 13.34% GOB_DIS_T: 21% GOB_CONT: 14% GOB_CONT_T: 13.66% UTLT: 13.34% GOB_CONT_C: 12.34% UTLT_C: 12%	GOB_DIS: 85% GOB_DIS_T: 70% GOB_CONT: 85% GOB_CONT_T: 70% UTLT: 85% GOB_CONT_C: 41.67% UTLT_C: 43.33%	GOB_DIS: 97.34% GOB_DIS_T: 97.34% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 97.34% GOB_CONT_C: 100% UTLT_C: 100%	GOB_DIS: 100% GOB_DIS_T: 100% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 100% GOB_CONT_C: 50% UTLT_C: 50%
1000	GOB_DIS: 13% GOB_DIS_T: 32.33% GOB_CONT: 13.66% GOB_CONT_T: 12% UTLT: 13% GOB_CONT_C11.67% UTLT_C: 12.34%	GOB_DIS: 81.66% GOB_DIS_T: 70% GOB_CONT: 78.34% GOB_CONT_T: 85% UTLT: 81.66% GOB_CONT_C: 31.67% UTLT_C: 46.67%	GOB_DIS: 98.8% GOB_DIS_T: 98.8% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 98.8% GOB_CONT_C: 100% UTLT_C: 100%	GOB_DIS: 50% GOB_DIS_T: 50% GOB_CONT: 100% GOB_CONT_T: 50% UTLT: 50% GOB_CONT_C: 50% UTLT_C: 0%
3000	GOB_DIS: 13.66% GOB_DIS_T: 34.33% GOB_CONT: 15.34% GOB_CONT_T: 13.34% UTLT: 13.66% GOB_CONT_C: 16.66% UTLT_C: 10.66%	GOB_DIS: 81.66% GOB_DIS_T: 50% GOB_CONT: 60% GOB_CONT_T: 81.66% UTLT: 45% GOB_CONT_C: 38.33% UTLT_C: 36.67%	GOB_DIS: 98.58% GOB_DIS_T: 98.58% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 98.58% GOB_CONT_C: 100% UTLT_C: 100%	GOB_DIS: 100% GOB_DIS_T: 100% GOB_CONT: 100% GOB_CONT_T: 100% UTLT: 100% GOB_CONT_C: 50% UTLT_C: 50%

Taula 2: Nivells de satisfacció dels objectius al llarg de l'execució.

Anàlisis dels resultats de la Taula 2:

- **Món Simple:**

  A les versions de GOB amb valors discrets (*GOB_DIS* i *GOB_DIS_T*) durant la simulació es manté sempre la mateixa tendència respecte a com evolucionen els objectius. Al cap de 3000 iteracions el *GoalSeguretat* i *GoalConstruir* acaben igual de satisfets (99.58% y 100% respectivament, uns valors molt bons). El *GoalAlimentar* acaba més satisfet en la versió sense temps, mentre que *GoalStocks* s'acaba més satisfet a la versió amb temps (degut a que les accions de recollir fusta i plantar vegetals són més curtes). Té sentit que a cadascuna acabi un objectiu més satisfet que a l'altre, per què la quantitat d'accions que inverteix *GOB_DIS* en satisfer el *GoalAlimentar*, *GOB_DIS_T* les inverteix en satisfer el *GoalStocks*.

  A les versions de GOB amb funcions contínues (*GOB_CONT* i *GOB_CONT_T*), al cap de les 3000 iteracions totes dues acaben amb una satisfacció completa per *GoalSeguretat* i *GoalConstruir*. En la versió sense temps (*GOB_CONT*) el *GoalStocks* queda lleugerament més satisfet, encara que molt similars, mentre que per la versió amb temps (GOB_CONT_T) ho fa el *GoalAlimentar*. El fet que *GoalStocks* acabi amb valors molt similars a l'estat final es deu a que s'utilitzen funcions contínues i no esglaonades (com passa a *GOB_DIS* i *GOB_DIS*), pel que en la mínima variació de la variable independent (número de pastanagues, fusta o herba), el valor de *discontentment* ja variarà per tant aquestes accions s'escullen més dinàmicament.

  En definitiva, es podria concloure que entre les versions de GOB el model *GOB_CONT_T* és més rentable d'utilitzar, ja que és el que té uns objectius amb uns valors de satisfacció més elevats.

  Pel que fa a la versió calculada amb el sistema d'utilitats (*UTLT*), veient els valors dels objectius al cap de 3000 iteracions es podria dir que és la millor aproximació, ja que *GoalSeguretat* i *GoalConstruir* queden completament satisfets (com en *GOB_CONT*), *GoalStocks* queda quasi satisfet (a diferència de les versions GOB) i *GoalAlimentar* és l'únic que quedaria amb uns valor més baix que les anteriors versions, encara que igualment pren el valor de 45%, que és un bon resultat.

- **Món Complex:**

  Pel que fa els models que utilitzen les variables introduïdes pel món complex, tant a *GOB_CONT_C* com a *UTLT_C* els valors de satisfacció finals són molt semblants. L'únic objectiu que queda completament satisfet és el de seguretat, *GoalConstruir* queda a la meitat i els dos objectius restants queden poc satisfets, pel que d'aquests models tampoc es poden extreure conclusions específiques.

  Hi ha un valor que crida l'atenció a *UTLT_C*: pel *GoalConstruir*, a la iteració 1000 es veu que acaba un 0% satisfet. No és tant estrany que passi per què com la tanca i el pont es van fent malbé amb el pas temps, pot haver coincidit

que just en les iteracions anteriors s'hagin destruït i que l'algorisme encara no hagi escollit l'acció pertanyent per tornar-los a construir.

7.1.3 Comparació de temps d'execució

El temps en segons que triga en executar-se per a cada versió dels algorismes es pot veure a la Taula 3.

Model	30 iteracions	1000 iteracions	3000 iteracions
GOB_DIS	0.004308701	0.02490425	0.04832649
GOB_DIS_T	0.010017400	0.03193283	0.06428146
GOB_CONT	0.004219055	0.02594185	0.05794144
GOB_CONT_T	0.008489609	0.02874565	0.06096458
GOB_CONT_C	0.006353378	0.01553345	0.03546143
UTLT	0.002529144	0.01522827	0.02988815
UTLT_C	0.007587433	0.01332664	0.02800369

Taula 3: Temps d'execució de cada model (en segons).

S'aprecia clarament com els temps d'execució de les versions de GOB són majors als del Sistema d'Utilitats, sobretot a mesura que s'incrementa el número d'iteracions.

En vista dels anàlisis de totes les versions, els objectius queden millor satisfets tant a *GOB_CONT_T* com a *UTLT*, i tenint en compte aquests temps d'execució, la tècnica que sembla més apropiada per implementar a en Lord Barus a Fracsland és *UTLT*, el sistema d'utilitats, ja que només tenint en compte les variables que afecten a cada acció s'escullen de forma compensada i els objectius queden ben satisfets amb el menor temps d'execució.

7.1.4 Comparació d'accions en un temps determinat

Una altra manera d'enfocar aquestes anàlisis i entendre millor el funcionament dels algorismes surt del fet de pensar que cada acció triga un temps diferent en fer-se, per tant, si les versions que tenen en compte el temps escullen accions de més curta durada que la resta de versions, les 30, 100 o 1000 iteracions trigaran menys temps en acabar-se.

Així, també s'ha analitzat l'estat del món i dels objectius observant el nombre d'accions que es fan en un temps determinat. S'ha quantificat aquesta durada atorgant a cada acció quantes iteracions trigaria en realitzar-se:

- Recollir fusta: 1 iteració.
- Plantar Herba, Plantar Pastanaga, Derrotar Lleó: 2 iteracions.
- Alimentar al Pollastre, Alimentar la Vaca, Alimentar el Conill: 3 iteracions.

- Construir Pont, Construir Tanca: 4 iteracions.
- Derrotar Capità Pirata: 5 iteracions.

D'aquesta manera, si les versions que no tenen en compte el temps escullen accions més llargues, amb el mateix nombre d'iteracions podran realitzar un nombre més reduït d'accions que les versions que sí que tenen en compte el temps. A la Figura 11 es veu un exemple d'aquest cas: cada fila de la taula és una unitat de temps d'iteració. Si per exemple *Feed Cow* triga tres iteracions en completar-se, en el mateix temps es podrien realitzar les accions *Plant Carrot* i *Pick Wood*, que triguen dues i una iteració en completar-se respectivament.

Versió sense temps	Versió amb temps
Plant Carrot	Build Fence
Defeat Pirate	
	Defeat Lion
	Pick Wood
Feed Cow	Plant Carrot
	Pick Wood

Figura 11: Comparació del nombre d'accions que es poden fer.

Aquesta comparació només té sentit fer-la per la tècnica GOB, que és a la que s'ha aplicat el factor temps en el seu algorisme. A la Taula 4 es mostra el nombre d'accions que es fan segons les iteracions amb les que s'executa l'algorisme.

Iteracions	**GOB_DIS**	**GOB_DIS_T**	**GOB_CONT**	**GOB_CONT_T**
20	13	13	13	13
500	194	225	171	182
3000	1157	1335	1002	1089

Taula 4: Nombre d'accions que es fan en un nombre determinat d'iteracions.

Es pot veure en la Taula 4 una clara diferència en el nombre d'accions que es fan en els models que tenen en compte el temps respecte les que no a mesura que s'incrementa el número d'iteracions. Això es deu a que aquestes dues versions (*GOB_DIS_T* i *GOB_CONT_T*) donen prioritat a les accions de curta durada, per tant en poden fer més en el mateix temps d'execució.

Amb aquest nou tipus de simulació, l'estat del món al cap de varies iteracions queda tal com es veu a la Taula 5. Inicialment els objectius segueixen tenint els mateixos nivells de satisfacció que en les simulacions explicades anteriorment.

A partir de l'anàlisi de la Taula 5 es poden extreure algunes conclusions:

- Pel que fa a els models que utilitzen valors discrets, s'extreuen millors resultats a *GOB_DIS*. Això es deu que *GOB_DIS_T* dona massa prioritat a les accions curtes i els objectius no acaben de quedar ben satisfets ni equilibrats del tot.
- Pel que fa als models que utilitzen valors donats per les funcions contínues, tots dos donen bons resultats i es compensa *GoalStocks* (que queda millor satisfet a *GOB_CONT*) amb *GoalAlimentar* (que ho fa a *GOB_CONT_T*).
- Entre les dues versions de GOB, *GOB_CONT* seguiria donant millors resultats per en Lord Barus, ja que gràcies a la utilització de funcions contínues evita canvis bruscs en els valors de *discontentment*.

Iteració	GOB_DIS	GOB_DIS_T	GOB_CONT	GOB_CONT_T
100	*GoalStocks*: 13.33% *GoalAlimentar*: 90% *GoalSeguretat*: 100% *GoalConstruir*: 50%	*GoalStocks*: 15% *GoalAlimentar*: 63.33% *GoalSeguretat*: 91.81% *GoalConstruir*: 50%	*GoalStocks*: 15.33% *GoalAlimentar*: 80% *GoalSeguretat*: 100% *GoalConstruir*: 100%	*GoalStocks*: 13.33% *GoalAlimentar*: 83.33% *GoalSeguretat*: 100% *GoalConstruir*: 100%
500	*GoalStocks*: 33.33% *GoalAlimentar*: 85% *GoalSeguretat*: 100% *GoalConstruir*: 50%	*GoalStocks*: 18.33% *GoalAlimentar*: 58.33% *GoalSeguretat*: 97.72% *GoalConstruir*: 100%	*GoalStocks*: 15% *GoalAlimentar*: 80% *GoalSeguretat*: 100% *GoalConstruir*: 100%	*GoalStocks*: 12.67% *GoalAlimentar*: 80% *GoalSeguretat*: 100% *GoalConstruir*: 100%
1000	*GoalStocks*: 33.67% *GoalAlimentar*: 90% *GoalSeguretat*: 99.8% *GoalConstruir*: 0%	*GoalStocks*: 25.67% *GoalAlimentar*: 58.34% *GoalSeguretat*: 98.6% *GoalConstruir*: 50%	*GoalStocks*: 15% *GoalAlimentar*: 83.34% *GoalSeguretat*: 100% *GoalConstruir*: 100%	*GoalStocks*: 12.33% *GoalAlimentar*: 86.66% *GoalSeguretat*: 100% *GoalConstruir*: 100%

Taula 5: Nivell de satisfacció dels objectius al llarg del temps.

7.2 Resultats

Un cop estudiats els resultats de les simulacions, s'ha vist que la millor tècnica per prendre decisions en el exemple concret que s'estudia en aquest projecte és la que utilitza el sistema d'utilitats.

Aquesta és la tècnica que se li ha integrat a en Lord Barus al joc, utilitzant només les accions que estan actualment disponibles a Fracsland com a missió, és a dir Construir el Pont, Alimentar al Conill, Alimentar al Pollastre, Alimentar a la Vaca i Derrotar al Capità Pirata.

Per consegüent, quan s'interacciona amb en Lord Barus en el joc, ja utilitza el nou mètode per fer la presa de decisions de la millor missió a fer i la mostra per pantalla per a que el jugador pugui veure la seva informació i acceptar-la.

Figura 12: Captura del joc on es pot veure el moment en que el jugador parla amb en Lord Barus i ell li mostra la missió a fer.

Un cop el jugador completa la missió que li ha assignat, pot tornar a parlar amb en Lord Barus per recollir el premi, que solen ser monedes o diamants, i automàticament li apareixerà la següent missió que ha escollit per a que accepti a continuació.

8 Conclusions i feina futura

8.1 Conclusions

Tal com s'ha mencionat al Capítol 2 (Objectius) d'aquest treball, l'objectiu principal d'aquest projecte era dotar a un personatge de Fracsland amb Intel·ligència Artificial per fer-lo més realista, així com estudiar varies tècniques per veure amb quina s'obtenien uns resultats més adients.

Amb aquest projecte s'han assolit aquests objectius, més concretament:

1. S'han estudiat varis models i algorismes per la presa de decisions basada en els objectius i accions d'un personatge.
2. S'ha estudiat el model per la presa de decisions basada en assignar valors d'utilitat a les accions possibles d'un personatge.
3. S'han implementat els algorismes amb l'entorn de Fracsland i s'han fet simulacions per a obtenir diferents resultats.
4. S'han analitzat els resultats obtinguts de la simulació per acabar d'estudiar les tècniques i veure quina funcionava millor en el cas concret de Fracsland.
5. S'ha implementat la millor tècnica obtinguda per el NPC escollit, en Lord Barus, per a que assignés missions al jugador segons l'estat actual del joc.

En general es pot concloure que s'han assolit els objectius plantejats a l'inici del treball i amb resultats satisfactoris, ja s'ha pogut trobar una bona tècnica d'Intel·ligència Artificial per aplicar al personatge seleccionat i tots els resultats obtinguts de les anàlisis fetes pels diferents algorismes han donat resultats coherents.

Tot i això, s'ha de tenir en compte les limitacions del temps inherents al projecte. No s'ha pogut acabar d'implementar totes les noves missions a Fracsland, aquesta part s'ha quedat només a les simulacions. La implementació que s'ha aconseguit fer a Fracsland és senzilla, només contempla les missions que existien prèviament. No obstant, s'ha aconseguit dotar al videojoc amb més dinamisme a l'hora d'assignar missions, una de les parts més importants del joc, i per tant s'ha aconseguit fer-lo més jugable.

També cal de destacar que aquest tipus d'algorismes requereixen invertir una gran quantitat de temps per fer proves i *debugging*. No hi ha cap solució única i correcta a l'hora de definir els valors de *discontentment* i utilitats, per tant s'ha d'invertir bona part del temps en fer *trial and error*: anar provant valors diferents fins a aconseguir uns resultats coherents.

8.2 Línies futures

Fracsland és un joc molt ampli que ja oferia un gran ventall de possibilitats sobre com estendre'l, i després de la realització d'aquest treball on se li ha introduït Intel·ligència Artificial, dona lloc a moltes altres línies futures diverses per a seguir ampliant el joc.

Seguint amb el tema principal en que s'ha centrat aquest projecte, es podrien acabar d'implementar totes les accions com a missions per a que en Lord Barus les pogués assignar al jugador, sobretot les relacionades amb tenir reserves d'aliments i materials, que actualment no es contemplen.

També es podria estendre la tècnica basada en el sistema d'utilitats, ja que en aquest treball s'ha implementat una primera aproximació, però en el futur es podria introduir el concepte d'utilitat marginal [11] per fer-la una mica més complexa.

De la mateixa manera, es podria implementar tot el model de món complex plantejat i crear més variables d'estat del món, més accions i més objectius per fer el joc una mica més complet i tenir més dinamisme en la presa de decisions d'en Lord Barus.

Un altre enfocament possible seria considerar el fet que poden haver diferents tipus de jugadors. Cada persona té un nivell d'experiència diferent i això comporta que a l'hora de realitzar les missions les pot completar amb un primer intent, o bé equivocar-se i haver-les de repetir varis cops. Això també podria influir en la presa de decisions del Lord Barus quan contempla el temps que triga en dur-se a terme cada acció.

Una altra extensió diferent de Fracsland seria fer-lo multijugador. Partint d'aquesta base, l'assignació de missions d'en Lord Barus podria modificar-se per donar diferents tipus de missions a cada equip o jugador diferent. Per exemple, fer que cada equip s'encarregués de realitzar les missions d'una zona del joc concreta.

Una altre línia de treball possible seria dotar d'Intel·ligència Artificial a altres NPCs del joc. Com es va esmentar a la Secció 4.2 d'aquest projecte, el venedor d'armes també era un personatge prometedor. Actualment deixa escollir al jugador l'arma que vol comprar, però per exemple podria recomanar-li la millor arma segons el seu nivell d'experiència o a la missió que s'ha d'enfrontar.

Seguint amb la línia de modificar altres NPCs, també es podria considerar la idea que en Lord Barus no només assignés missions al jugador, sinó també a altres NPCs (com els vilatans del poble) per a gestionar l'illa d'una manera més òptima.

Apèndix A: Manual tècnic

En aquesta secció es detallen els passos necessaris a seguir per poder provar i desenvolupar Fracsland.

A.1 Instal·lació: Requeriments mínims i passos a seguir

A.1.1 Requeriments mínims

Per poder desenvolupar aquest projecte s'han de complir uns requeriments mínims que venen establerts per Unity.

A nivell de software requereix que el sistema operatiu sigui Windows 7, 8 o 10 versió de 64 bits o bé Mac OS X 10.9 cap endavant.

En quant a hardware requereix una targeta de vídeo amb capacitat per DX10 (shader model 4.0) i que la CPU sigui compatible amb el conjunt d'instruccions SSE2.

Es pot trobar més informació sobre els requeriments de Unity per a desenvolupadors a la següent adreça: https://unity3d.com/es/unity/system-requirements

Si només es vol executar el joc, els requeriments són els descrits prèviament a la Secció 4.4 (Requeriments Tecnològics).

Addicionalment, aquest projecte és molt extens i complex: compta amb una gran quantitat de *Assets* que fan que ocupi molt d'espai, pel que també és necessari tenir mínim 3GB de memòria disponibles a l'ordinador per a poder descarregar-lo.

A.1.2 Instal·lació del Servidor

Cada cop que es vulgui executar el joc de Fracsland, és necessari que el servidor estigui executant-se per a que li pugui proveir al joc amb els JSONS que necessiti en cada moment.

És un projecte de *Netbeans*. S'ha de configurar el primer cop abans d'executar-lo. S'ha d'obrir la classe `FracslandServerFake.java` i modificar la variable `dir`. Allà s'ha d'introduir la ruta absoluta de la carpeta on es troba aquesta classe, per exemple: `/Users/aina/FracslandServerFake/src/fracslandserverfake`

A.1.3 Instal·lació de Fracsland

Per poder desenvolupar Fracsland s'ha de descarregar i instal·lar *Unity3D* i *VisualStudio* o *Monodevelop* per poder editar el codi C#.

Per assegurar-se de que no hi hagi problemes de compatibilitat amb diferents versions de Unity, es recomana descarregar-se la versió *2017.3.0.p3*. Es pot trobar a l'arxiu de versions de Unity a la seva pròpia pàgina web:

`https://unity3d.com/es/get-unity/download/archive`

El projecte conté dues carpetes: Simulacions i Joc. Totes dues contenen el codi del projecte, però en la de Simulacions, quan s'executa el joc i s'interactua amb en Lord Barus, no s'executa la nova lògica implementada però per Logs es pot veure els resultats de la simulació amb X iteracions. En canvi, en la carpeta de Joc, conté el codi que executa el joc normalment incloent a en Lord Barus amb la seva nova lògica implementada.

Un cop descarregat, es pot importar el projecte de la carpeta desitjada a Unity i s'hauran de fer algunes modificacions en el codi per enllaçar-lo amb el servidor abans de poder executar el joc per primer cop. A la classe `Constants.cs`, modificar la constant `API_BASE_URL` amb la mateixa ruta absoluta especificada a la variable `dir` del Servidor a la classe *FracslandServerFake.java*, però afegint al final "`/api/v1/`". Per exemple:

`/Users/aina/FracslandServerFake/src/fracslandserverfake/api/v1/`

Finalment, s'ha d'accedir a l'escena de Login per poder executar el joc correctament. Totes les escenes es troben al directori del projecte Unity: *Assets >Resources >Scenes.*

A.2. Manual del desenvolupador

En aquest apartat es detallen alguns aspectes per a poder modificar el codi introduït en aquest treball al joc de Fracsland. El projecte està estructurat en dues carpetes: Joc i Simulacions. La carpeta Joc conté el codi de tot el projecte amb els scripts preparats per a que quan s'executi el Lord Barus ja assigni missions amb la nova IA implementada. La carpeta Simulacions conté la carpeta dels *Scripts* que s'haurien de posar al projecte per a que executi el joc i mostri els resultats de les simulacions per la consola de Unity. Es pot veure la jerarquia de carpetes a la Figura 13.

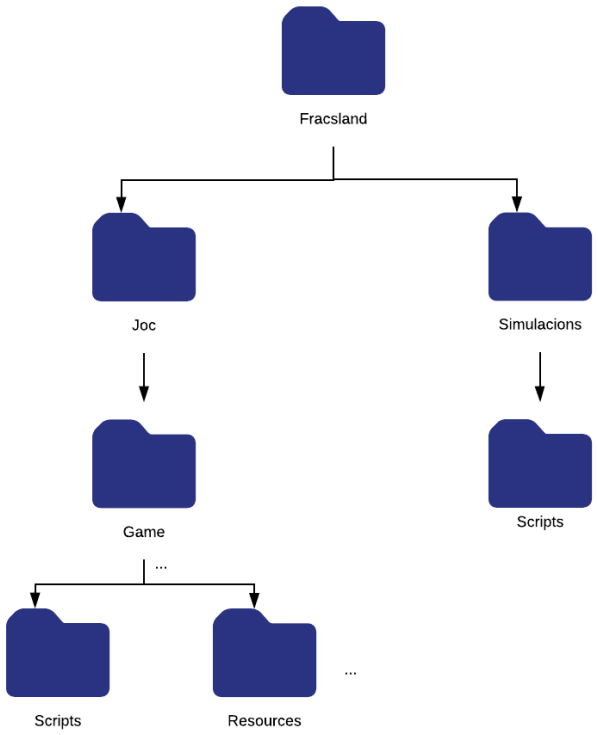


Figura 13: Estructura del projecte.

A.2.2 Joc

Aquesta carpeta conté el codi del projecte amb la nova lògica de presa de decisions implementada pel Lord Barus.

En cas de voler estendre o canviar aquesta lògica, s'ha d'anar a la classe que es troba en el fitxer `MissionSelectorMenu.cs`. Conté un mètode anomenat `loadMissionsIntoUI`, que s'encarrega d'escollir una missió per al jugador i mostrar-la per pantalla. Inicialment cridava al mètode `chooseRandomQuests` de la classe `QuestManager`, que tal com indica el seu nom, agafava de forma aleatòria una missió de cada zona de Fracsland i la retornava en una llista. Actualment, en la mateixa classe s'ha implementat un nou mètode anomenat `chooseBestQuest` que és l'encarregar d'executar l'algorisme de presa de decisions i retornar una única missió.

A.2.1 Simulacions

Aquest directori conté una carpeta *Scripts* amb el codi preparat per a que es mostri per la consola de Unity els resultats de la simulació amb un nombre determinat d'iteracions.

Per executar el joc en mode simulació, s'ha de canviar la carpeta *Scripts* que té el projecte (a Joc, preparat amb la lògica d'en Lord Barus), per la carpeta *Scripts*

que hi ha a Simulacions.

Totes les classes que s'han creat en aquest projecte es troben en el repertori *GameStats* dins de la carpeta *Scripts*. En cas de voler modificar el funcionament de les simulacions, es pot accedir a la classe `ActionSimulator`. És la classe principal que ho gestiona tot.

Actualment el mètode de presa de decisions que s'executa al jugar a Fracsland és el Sistema d'Utilitats, però també estan implementats tots els altres mètodes explicats en aquest treball. Els passos a seguir per executar el joc amb les diferents tècniques són els següents:

- GOB

  Per utilitzar el joc amb els diferents algorismes de GOB, a la classe `ActionSimulator`, el mètode `simulateActionChooser` ha d'utilitzar el mètode `chooseGOB`.

  Per a provar-los només s'han de modificar un seguit de constants que es troben a la classe `Constants.cs`:

  - `DISCRETE_FUNCTION` : Pot valer 0 o 1. Si val 0, l'algorisme de GOB s'executarà utilitzant valors discrets per calcular el nivell de *discontentment*. Si val 1 s'executarà utilitzant els valors que donen les funcions contínues implementades.
  - `CALCULATE_DURATION` : Pot valer 0 o 1. Si val 0, l'algorisme de GOB s'executarà sense tenir en compte el temps. Si val 1 sí que calcula els valors de *discontentment* tenint en compte la durada de les accions.

- Utility System

  Per utilitzar el joc amb utilitats, a la classe `ActionSimulator` el mètode `simulateActionChooser` ha de fer servir el mètode `chooseUtilities`.

  A més de fer la simulació amb les variables que conté el món actual a Fracsland, també pot fer-se tenint en compte les noves introduïdes per fer els factors de decisions més elaborats. A la classe `Constants.cs` s'ha de modificar la constant `WORLD_SIMPLE_COMPLEX`, sent 0 si es vol jugar amb el món "simple"(utilitzant les variables del joc) o 1 si es vol que també es tinguin en compte les noves.

Referències

[1] Muriel Ordóñez, C. *Fracsland: joc seriós per aprendre fraccions.*, TFG, 2016.

[2] Orkin, J., *Applying Goal-Oriented Action Planning to Games.*, Productions, 2003.

[3] Susi, T., Johannesson, M., & Backlund, P., *Serious games: An overview.*, Institutionen för kommunikation och information, 2007.

[4] Rasmussen J., *Are Behavior Trees a Thing of the Past?*, 2016. Últim accés: 26 de juny de 2018 des de https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/

[5] Millington, I. & Funge J., Decision Making, *Artificial Intelligence For Games*, 401-413, Bullington, Morgan Kaufmann Publishers, 2009.

[6] Champandard A., *Living with The Sims AI*, 2007. Últim accés: 26 de juny de 2018 des de http://aigamedev.com/open/highlights/the-sims-ai/

[7] Palacios, J., Decision Making, *Unity 5.x Game AI Programming Cookbook*, 111-113, Packt Publishing, 2016.

[8] Alastair A., *At-a-glance functions for modelling utility-based game AI*, 2013. Últim accés: 23 de juny de 2018 des de https://alastaira.wordpress.com/2013/01/25/at-a-glance-functions-for-modelling-utility-based-game-ai/

[9] Graham, D., An Introduction to Utility Theory, *Game AI Pro*, 113-126, CRC Press, 2013

[10] *DecisionFlex*, 2014. Últim accés: 26 de juny de 2018 des de https://assetstore.unity.com/packages/tools/ai/decisionflex-8967

[11] Mark, D., The Concept of Utility, *Behavioral Mathematics for Game AI*, 111-129, Boston, MA USA : Charles River Media, 2009.